digital

TB/CACHE/SBI CONTROL TECHNICAL DESCRIPTION

VAX 11 780

BLANK

EK-MM780-TD-001

Translation Buffer, Cache and SBI Control Technical Description (VAX-11/780 Implementation)

digital equipment corporation • maynard, massachusetts

First Edition, April 1978

Printed in U.S.A.

This document was set on DIGITAL's DECset-8000 computerized typesetting system.

CONTENTS

CONTENTS (Cont)

CONTENTS (Cont)

FIGURES

FIGURES (Cont)

FIGURES (Cont)

TABLES

TABLES (Cont)

CHAPTER 1
INTRODUCTION

1.1 MANUAL SCOPE

This manual provides a comprehensive description of the functional and operational characteristics of the Translation Buffer, Cache, and SBI Control of the VAX-11/780 system. The manual is written at two levels of detail to provide a resource for appropriate branch and support level courses of the Field Service training program and to provide a field reference. The two levels are:

1. Introduction

2. Functional/Logic Description.

The introduction gives a brief description of the architecture of the Translation Buffer, Cache, and SBI Control and a simplified explanation of their functions. It also contains an explanation of the memory management scheme, defining the virtual-to-physical address relationship.

The functional/logic description provides a more detailed explanation of the operational characteristics of each device. It also contains a detailed explanation of address translation.

Table 1-1 is a list of related hardware manuals and their availability.

1.2 OVERVIEW

In the VAX-11/780 system, over four billion bytes of virtual memory space are mapped to over one billion bytes of physical memory space. Virtual-to-physical address translations are performed by the CPU microcode. The Translation Buffer provides a buffer store for translations and associated memory protection information. In addition the VAX-11/780 system provides a Cache to reduce the average memory access time. The SBI Control provides an interface between the CPU system and the SBI.

Figure 1-1 illustrates these three functional areas and their interconnection in the CPU:

1. *The Translation Buffer* is a two-way set associative buffer which stores virtual-to-physical address translations and corresponding memory access protection information. The addressing and protection data contained in the Translation Buffer (TB) is only for the process currently executing on the system. In the TB a virtual address (from an address source in the data path) selects the correct physical address for reference. If the TB does not contain the translation, the firmware performs the translation and the results are placed in the TB for future use.

2. *Cache* is a two-way set associative buffer for the storage of data which will most likely be required by the process(es) currently executing on the system. In Cache a physical address is compared against a stored address to select the correct data. If Cache does not contain the data, it is fetched from memory and placed in Cache for possible reuse. Data in Cache may be accessed in 200 ns to speed system operations.

3. *SBI Control* is the CPU interface to the SBI (Synchronous Backplane Interconnect). The SBI Control transfers data between the SBI and the internal components of the CPU system.

Table 1-1 Related Hardware Manuals

Title	Document Number	Comment
VAX-11 KA780 Central Processor Technical Description	EK-KA780-TD-PRE	In microfiche library*
VAX-11 MS780 Memory System Technical Description	EK-MS780-TD-PRE	In microfiche library*
VAX-11 DW780 Unibus Adaptor Technical Description	EK-DW780-TD-PRE	In microfiche library*
VAX-11/780 Console Interface Board Technical Description	EK-KC780-TD-PRE	In microfiche library*
VAX-11/780 Architecture Handbook	EB 07466	Available on hard copy†

*For information concerning microfiche libraries, contact:

Digital Equipment Corporation
Micropublishing Group (PK3-2/T12)
129 Parker Street
Maynard, MA 01754

†This document can be ordered from:

Digital Equipment Corporation
444 Whitney Street
Northboro, MA 01532
Attention: Communication Services (NR2/M15)
Customer Services Section

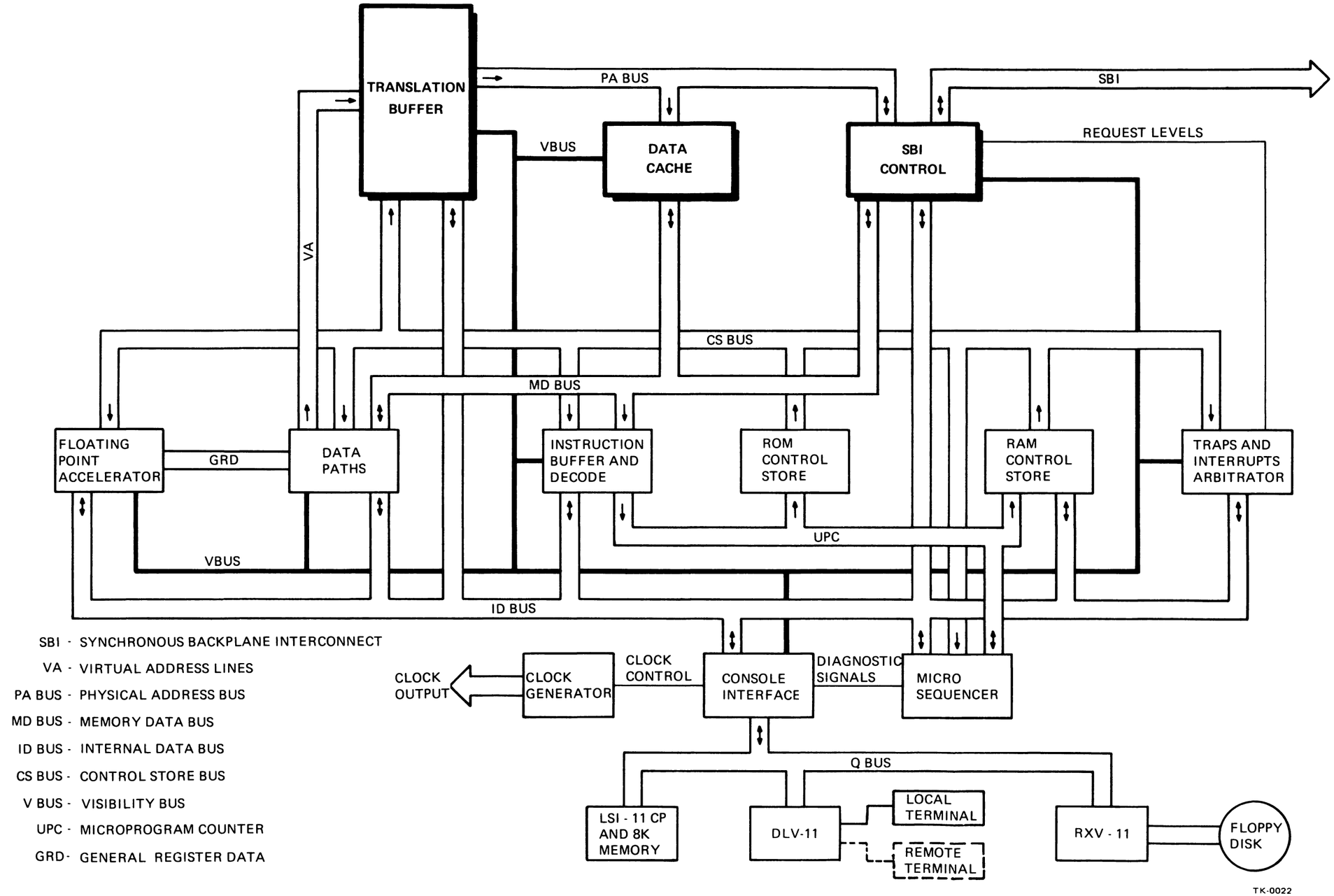


Figure 1-1 CPU Block Diagram

1.3 ADDRESS SPACE

VAX-11/780 employs a memory management scheme to map the user's virtual space of 4 billion bytes to a physical space of 1 billion bytes. (Memory management also provides protection.) When a process is executing on the system, some of its pages are in memory while others may be resident on disk. The system software ensures the correct information is transferred from disk to main memory as required for program execution (Figure 1-2). The address space available to a process is a linear array of 4 billion bytes.

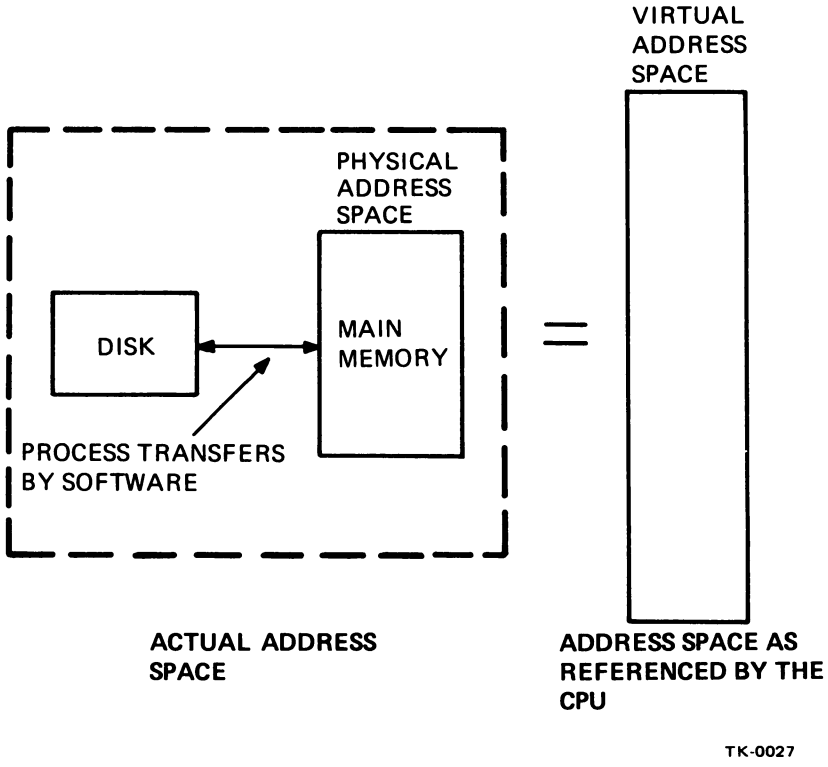


Figure 1-2 Virtual/Physical Relation

1.3.1 Virtual Space

The virtual address space (Figure 1-3) is defined by a 32-bit address and extends from address 0 to address $FFFFFFFF_{16}$. It has two parts. The lower half (per process space) is distinct for each process executing on the system and extends from address 0 to address $7FFFFFFF_{16}$. A context switch changes the mapping of all locations in process space to accommodate the current process (Paragraph 1.6). The upper half (system space) is shared by all processes (remains the same during context switching) and extends from address 80000000_{16} to address $BFFFFFFF_{16}$. Note that the highest quarter of virtual space (C0000000-$FFFFFFFF_{16}$) is reserved and not currently used.

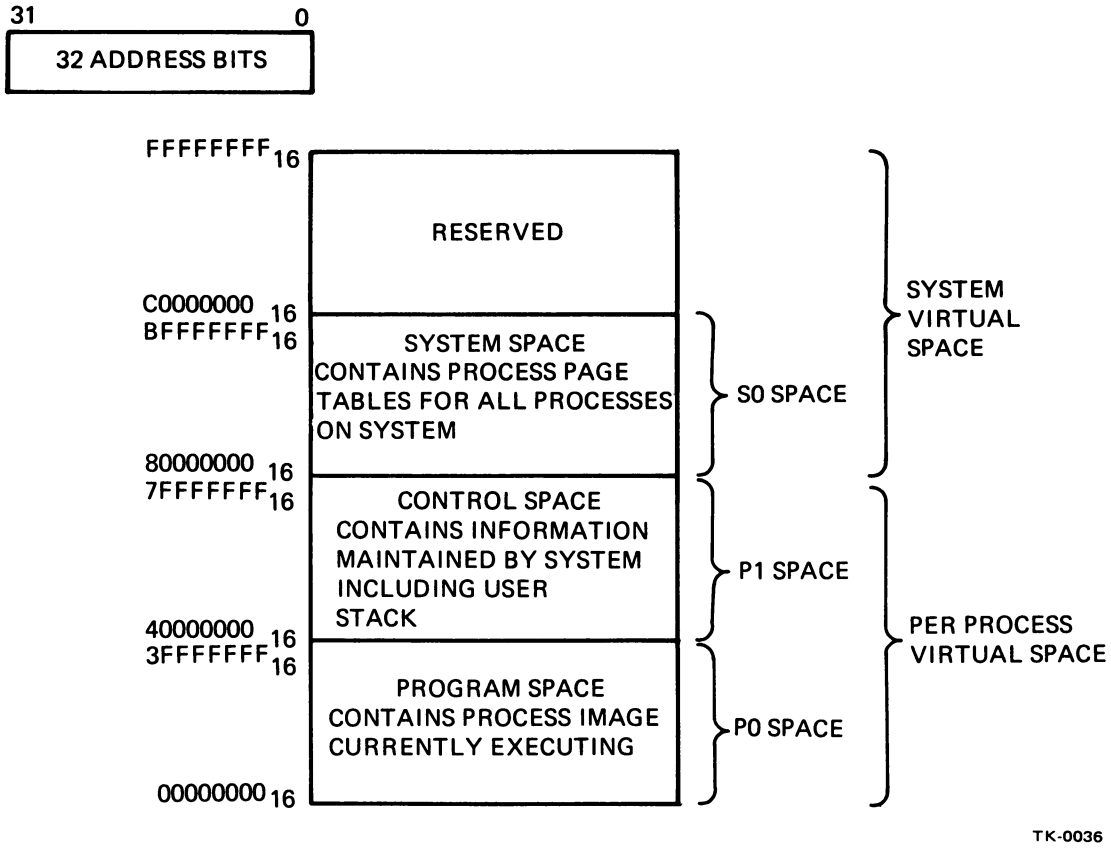


Figure 1-3 Virtual Address Space

Furthermore, the per process space has two sections. The program space (P0) occupies the lower half of the process space and contains process image(s) currently executing on the system. The control space (P1) occupies the upper half of the process space and contains user stacks and I/O buffers for the process currently executing on the system. The distinction between upper and lower process space allows software to allocate one section for programs which grow from lower to higher addresses and one section for user stacks which grow from higher to lower addresses.

1.3.2 Physical Space

Each virtual address generated by the CPU must be translated into a physical address to locate the information in main memory. The translated physical address corresponds to an actual memory location.

As shown in Figure 1-4, the physical address space is defined by a 30-bit address (1 billion bytes) and extends from address 0 to address $3FFFFFFF_{16}$. The physical space has two parts. Device control registers are assigned to the upper half which extends from address 20000000_{16} to $3FFFFFFF_{16}$. The lower half is reserved for primary memory and extends from address 0 to $1FFFFFFF_{16}$.

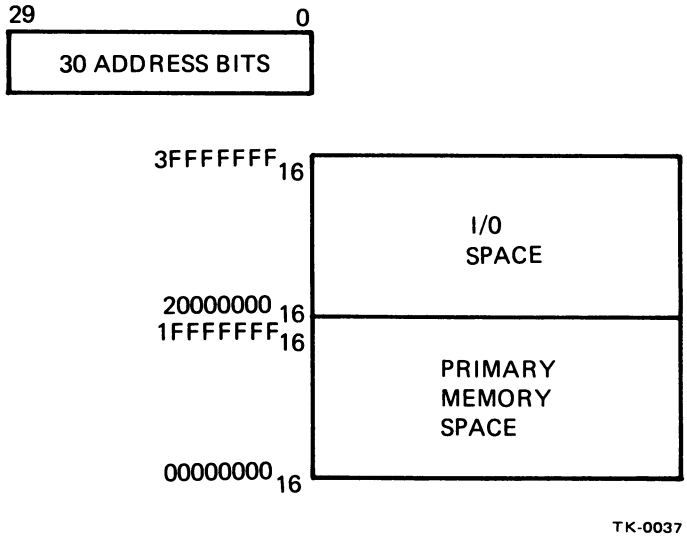


Figure 1-4 Physical Address Space

1.4 VIRTUAL PAGES AND PHYSICAL FRAMES

The entire virtual address space is divided into 512-byte pages whose boundaries are invisible to the executing process(es). As seen in Figure 1-5, each region of the virtual address space (system, program, and control) is divided into these virtual pages and numbered contiguously. Likewise the memory address space is divided into blocks, each of which can contain one virtual page. Each block of memory designatable to a virtual page is called a physical page frame.

As in Figure 1-5, physical memory appears to contain disjointed virtual pages in contiguous page frames. Therefore, to reference this data, the virtual address is translated to a physical address which selects a page frame number and byte within the page.

The virtual address format is shown in Figure 1-6. As seen in this figure a virtual address selects a section of virtual address space (system control or program), a page within that section, and a byte within the page. The portion of the address which selects a byte within the page can be used without modification once the page frame number has been determined.

When system software relocates a page, a record of the page frame number is stored in memory. A collection of these records is kept for each region of virtual space. The collections are called page tables (paragraph 1.5). A unique pair of page tables map process space for each process. System space, however, is mapped by one page table for all processes.

1.5 PAGE TABLES

Maps of virtual page locations are stored in memory as page tables. Page tables are maps from virtual addresses to physical locations. Each entry on the page table (page table entry or PTE) contains translation and protection information for its corresponding virtual page (paragraph 1.5.1).

A separate page table is provided for each section of the virtual address space (i.e., a separate page table for system space, control space, and program space). Each process has a unique pair of page tables for the process space. This means that during a context switch, although the same system space page table remains in use, different program space and control space page tables become usable. Because all processes use the same system space map, this space is considered shared among all processes.

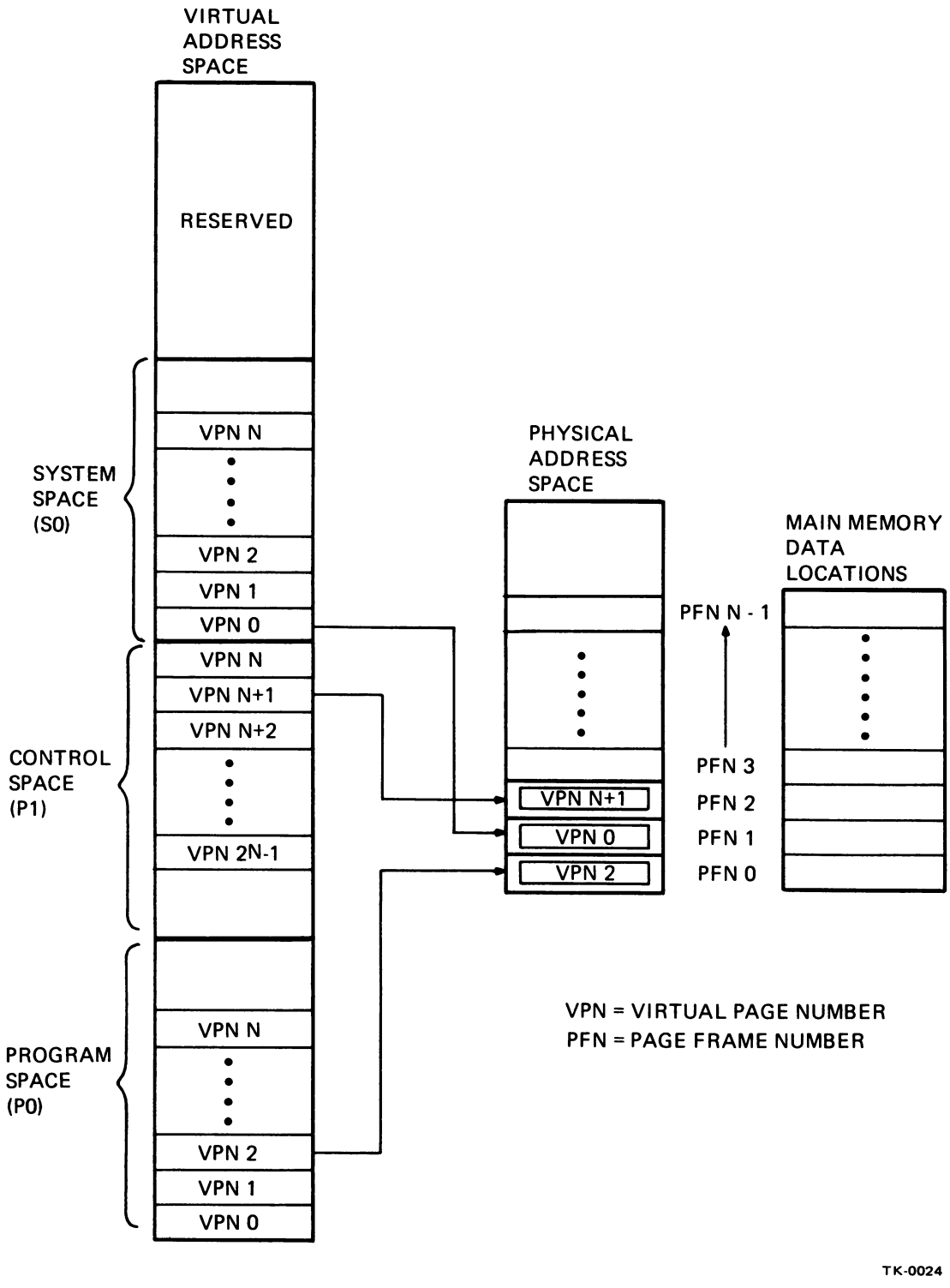


Figure 1-5 Examples of Page Frame Allocation

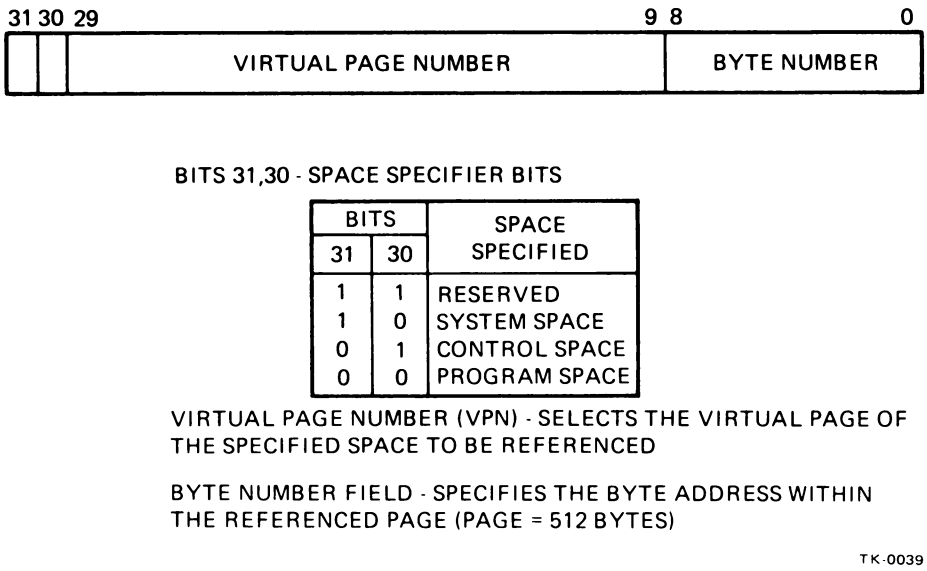


Figure 1-6 Virtual Address Format

Each page table is maintained by system software. Process page tables reside in (paged) system virtual space. The locations of the tables in physical memory are thus mapped by the system page table (Figure 1-7). The system page table always exists contiguously in physical space. Process page tables always exist in system virtual space but may not exist in physical memory (they may be out on disk).

The system page table maps (allocates) system virtual addresses to physical locations. Likewise, the process page tables map process virtual addresses to physical locations. System virtual addresses typically contain system data, where process virtual addresses contain program and control data.

Note that although the process page tables may exist on two or more contiguous virtual pages, their respective pages are mapped individually. Thus their pages may exist disjointly in physical space. The system page table, however, always exists in physical space contiguously, regardless of the number of pages.

1.5.1 Page Table Entry

Each entry in a page table contains translation and protection information for its corresponding virtual page. Figure 1-8 illustrates the format of a page table entry (PTE). As seen in this Figure, the page table entry contains the page frame number which is the physical address of the virtual page it represents. Together with the byte offset from the virtual address a physical reference can be made.

The page table entry also contains a protection field for the page. The protection code describes the read/write accessibility of the associated page for each of the processor modes. This is accomplished by comparing the protection code against the current mode of the processor (from the processor status longword, PSL) and the intended access (read, write, etc.).

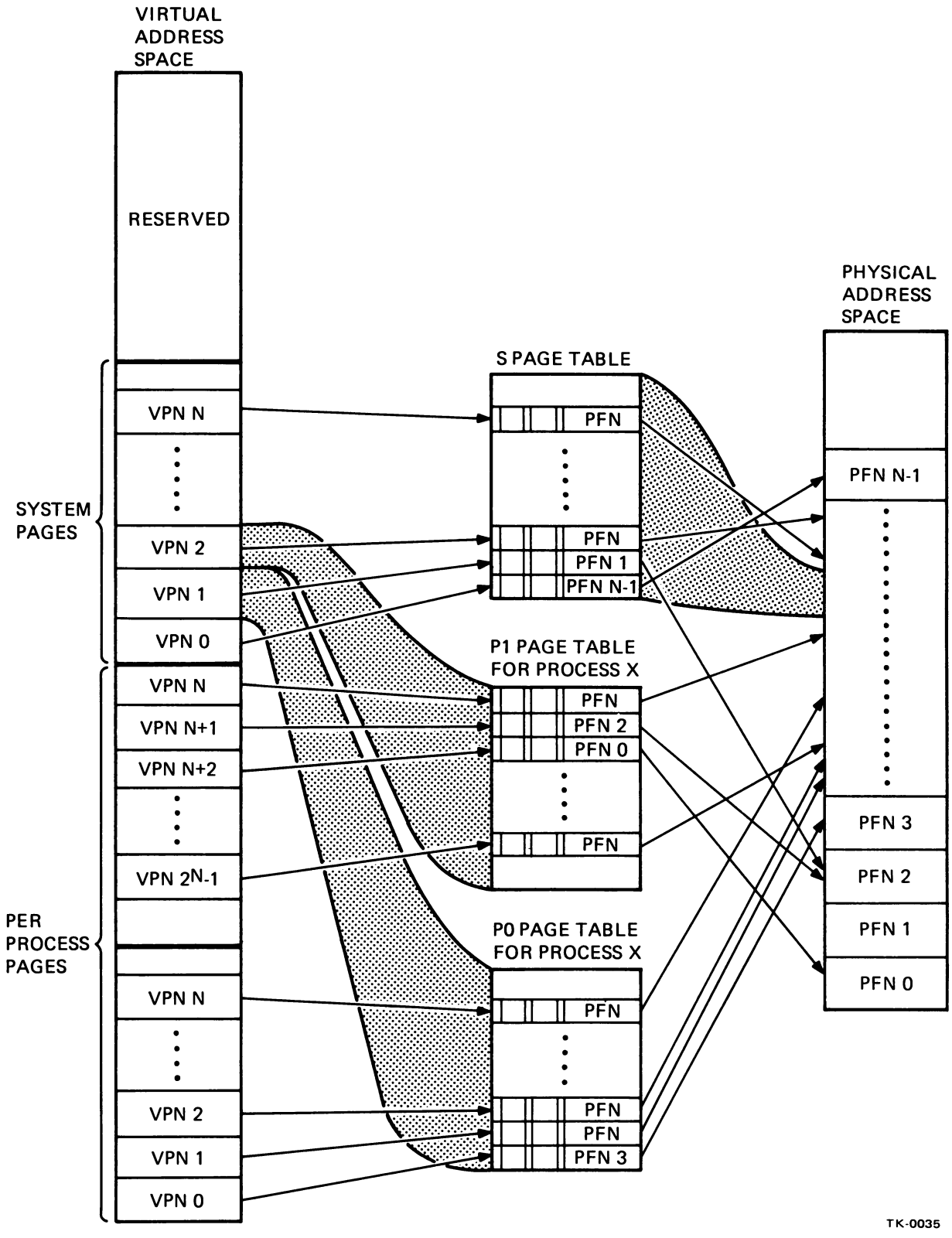
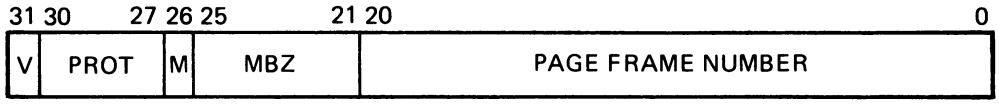


Figure 1-7 Virtual Pages Mapped to Physical Space



VALID BIT (V)	GOVERNS VALIDITY OF M BIT AND PFN FIELD V = 1; PAGE CAN BE ACCESSED BY EXECUTING PROCESS V =0; PAGE CANNOT BE ACCESSED BY EXECUTING PROCESS
PROTECTION FIELD (PROT)	ALWAYS VALID AND USED BY HARDWARE EVEN WHEN V = 0
MODIFY BIT (M)	M = 1 IF PAGE HAS ALREADY BEEN RECORDED AS MODIFIED M = 0 IF PAGE HAS NOT BEEN RECORDED AS MODIFIED USED BY HARDWARE ONLY IF V = 1
BITS<25:21>	USED BY SYSTEM SOFTWARE
PAGE FRAME NUMBER (PFN)	UPPER 21 PHYSICAL ADDRESS BITS OF THE PAGE LOCATION USED BY HARDWARE ONLY IF V = 1.

TK-0040

Figure 1-8 Page Table Entry Format

The processor modes, ordered from most privileged to least privileged, are:

1. *Kernel* – Used by the kernel of the operating system for I/O drivers and page fault handling.
2. *Executive* – Used for the majority of the operating system service calls.
3. *User* – Used for user level code, utilities, compliers, etc.

The mode at which the processor is currently running is stored in the current mode field of the PSL. If the processor mode and protection code for the page about to be accessed indicate that the access would be illegal, the page access is inhibited and an access control fault occurs.

As seen in Figure 1-8, a PTE also contains a valid (V) bit. For the PTE in main memory, the condition of the V bit is set by the operating system to indicate the status of the page. This condition is examined when a PTE is fetched from main memory during a TB miss. If the V bit is not set, a macrofault routine is executed. During the routine, the operating system finds the correct page table and updates the PTE by setting the V bit. The memory reference is then retried.

The V bit of a PTE in the TB indicates the validity of the PTE. If a PTE in the TB is accessed and the valid bit is not set, a TB miss microtrap occurs.

All PTEs in the TB are invalidated during system initialization. This provides invalid PTEs with good parity. Similarly during a context switch all process PTEs are invalidated. This is because the mapping of each process virtual page changes when a context switch occurs.

Also all PTEs contain a modify (M) bit. The M bit of a PTE in main memory is an indication to the operating system that the page has or has not been modified while in main memory. With this information the operating system can decide whether or not it must rewrite the page on disk when it elects to remove the page from main memory. If while in main memory the page is modified by the processor (written into), the page must be rewritten on disk when the operating system elects to remove it from main memory. Otherwise, without modification to the page, a rewrite to disk is not needed (i.e., the copy on disk continues to contain the most current copy of the page).

When a PTE is loaded into the TB from main memory, a copy of the M bit is included. Each time a write is executed, the M bit of the PTE in the TB is examined. If the M bit is not set, a microtrap occurs. During the M bit microtrap, the PTE from main memory is fetched, the M bit is set, and the PTE is rewritten to memory. This is an indication to the operating system that the page has been modified. Likewise the M bit of the PTE in the TB is asserted.

1.5.2 Base and Length Registers

The base and length registers are accessed during an address translation. The base registers indicate the location of the page tables for the executing process. The length registers provide information for a length violation check. Figure 1-9 illustrates their respective formats.

In all three base registers, bits 1 and 0 are always 00 indicating page tables are longword aligned. The contents of the process base registers are system virtual addresses (bit 30 and 31 are 10). The system base register contains a physical address.

Generally, a length register contains the length of a page table in longwords. However, the P1 length register contains the page table length of a page table describing the unused portion of P1 space. In other words, the P1 length register contains 2^{21} (number of virtual pages in the P1 space) minus the length of the P1 page table in longwords. This is due to the direction of growth in the P1 region.

1.6 PCB AND CONTEXT SWITCHING

A process is partially defined by the contents of its hardware process control block (HPCB). A HPCB is assigned to each process on the system. The HPCB contains all the switchable process context collected in a single contiguous block of address space for efficient transfer to/from the internal processor registers.

The major hardware-visible contents of the PCB are the contents of the general purpose registers, the P0 and P1 base and length registers, the program counter, and the processor status longword. Figure 1–10 illustrates the PCB contents.

All HPCBs reside in system virtual space. For a process to execute on the system, the PCB content must be loaded in the processor internal registers. This is accomplished by a software instruction (LDPCTX). To context switch, the HPCB of the executing process is stored in memory and a new HPCB is loaded in the processor internal registers. This permits the CPU to execute a number of processes in a timesharing scheme. Note that a context switch changes the mapping of all process virtual addresses, because new process page tables are defined by new base and length registers (Figure 1-11). As seen in this Figure, a reference to virtual page 1 will access the contents of page frame 5 during the execution of process A. A reference to the same virtual page during the execution of process B, however, will access the contents of page frame 1. The mapping of all other virtual pages in this example may likewise be dissimilar for each process.

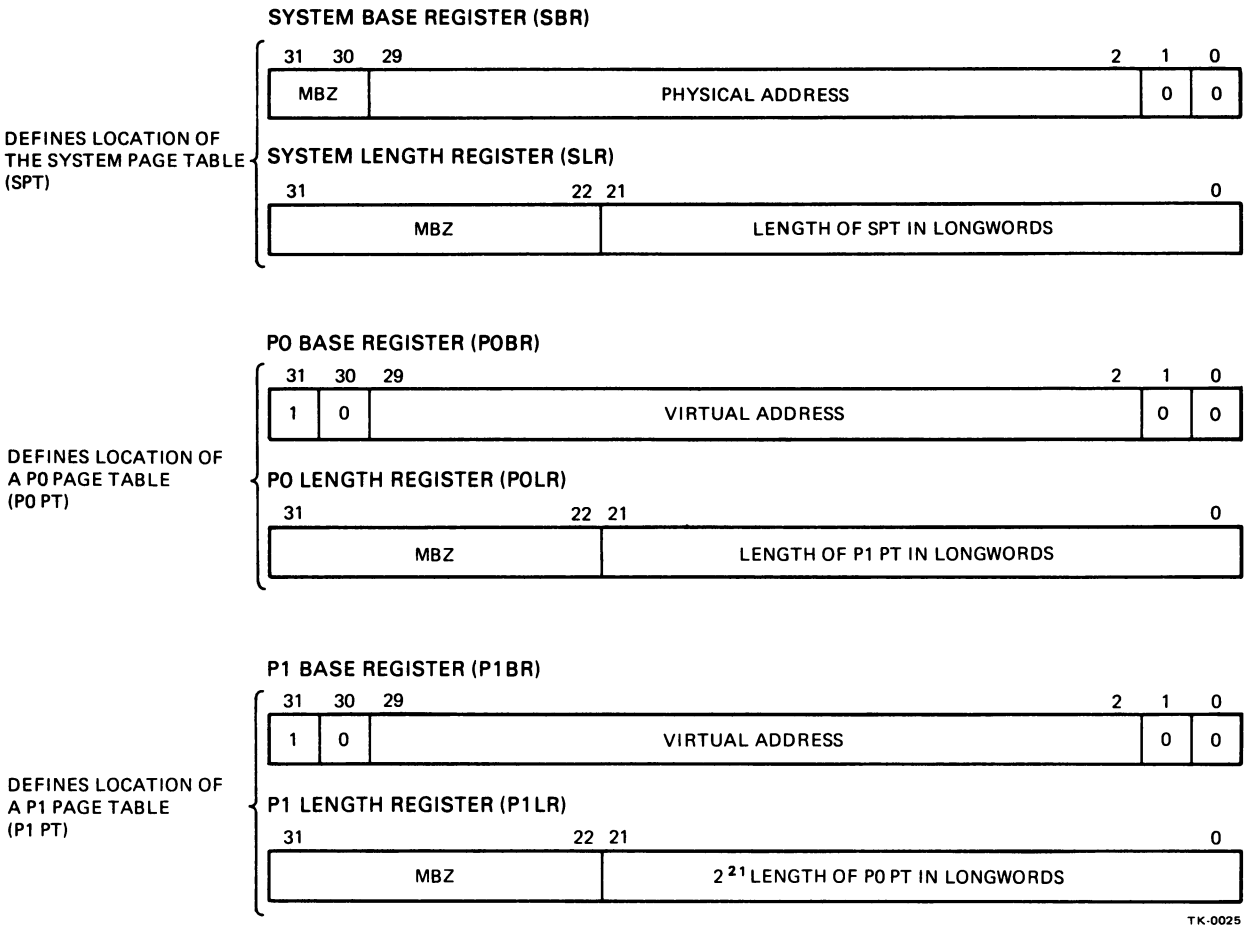


Figure 1-9 Base and Length Registers

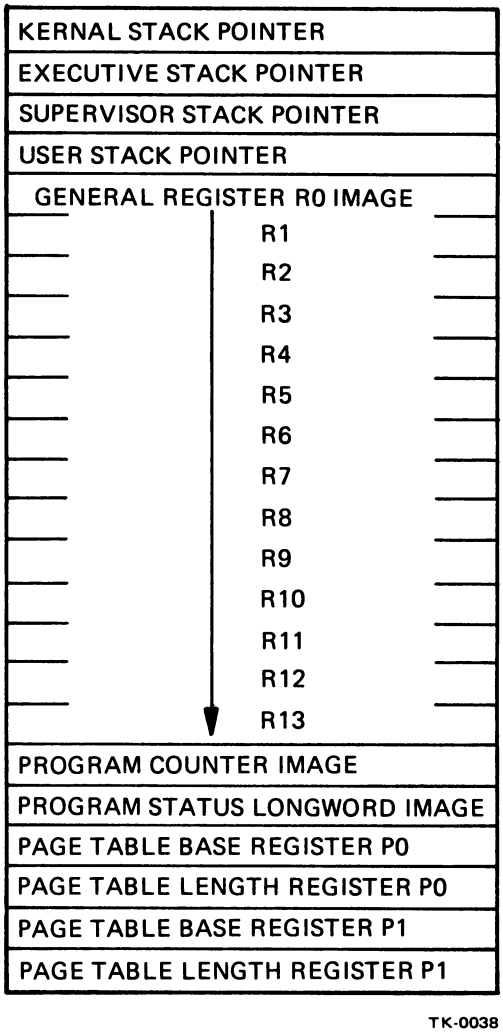


Figure 1-10 Hardware-Visible PCB Content (HPCB)

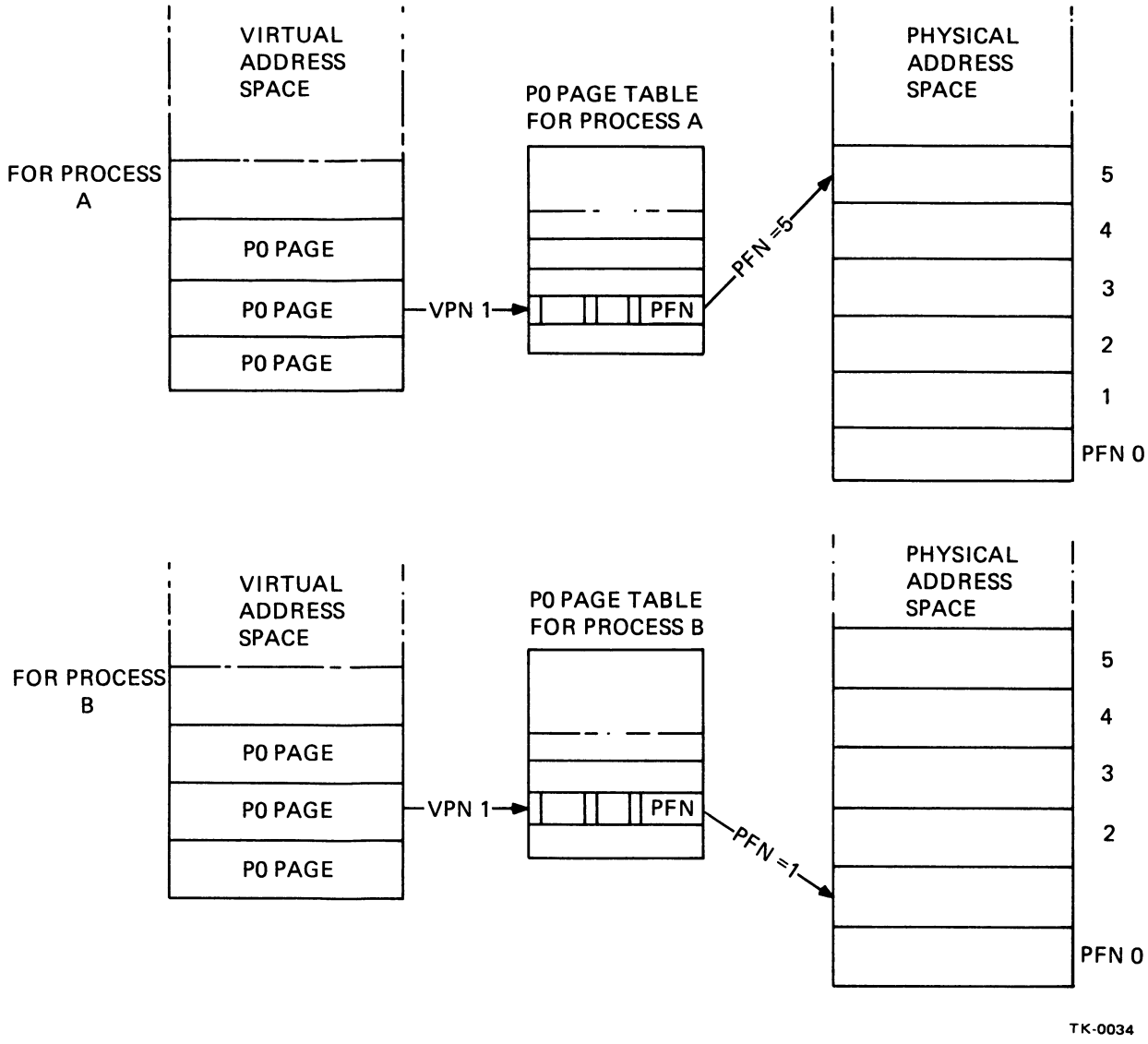


Figure 1-11 Example of Context Switching and Page Frame Allocation

1.7 ADDRESS TRANSLATION - INTRODUCTION

To locate information in physical memory, virtual addresses generated by the CPU must be translated to physical addresses. All address translations are actually performed by the CPU microcode. The TB merely stores the results of the translations to speed overall memory access time. Thus each time a TB miss occurs (the translation information is not in the TB), an address translation must be performed by the microcode.

The translation results stored in the TB are page table entries which contain protection and relocation information for the accessed virtual page. Address translation is further defined in the following section and the Functional/Logic Description.

1.7.1 Address Translation - Simplified Operational Description

This section provides a simplified description of the interaction of the Translation Buffer, Cache, and SBI Control. Memory access examples are examined and described in a simplified form for the sake of discussion. The Functional/Logic Description provides details.

These paragraphs describe the sequence of address translation for each of the following CPU memory access cases:

1. Reference to system virtual space with a TB hit
2. Reference to system virtual space with a TB miss
3. Reference to process virtual space with a TB hit
4. Reference to process virtual space with a single TB miss
5. Reference to process virtual space with a double TB miss.

With each description a corresponding figure is provided, illustrating the sequence.

1.7.1.1 Reference to System Virtual Space with a TB Hit (Figure 1-12, Part A)

1. The CPU data path generates a system virtual address and sends it to the TB. The address matches a virtual address in the TB (TB hit) which indicates the corresponding system page table entry (S PTE) is in the TB. Note that a TB hit also indicates the valid bit is set

2. The page table entry contains the upper bits of the physical address of the desired data. If the physical address is in Cache, the corresponding Cache data may be accessed. (A write to Cache results in a simultaneous update of main memory.) If the physical address is not in Cache, main memory must be accessed for the data.

1.7.1.2 Reference to System Virtual Space with a TB Miss (Figure 1-12, Part B)

1. The CPU data path generates a system virtual address and sends it to the TB. The address does not match a virtual address in the TB (TB miss), which indicates the corresponding system page table entry (S PTE) is not in the TB and must be fetched from main memory.

2. The CPU microcode calculates the physical address of the system page table entry.

3. The physical address locates the system page table entry in main memory (or Cache).

4. The system page table entry is retrieved from main memory and placed in the TB (provided the V bit is set). The corresponding addressing information (TAG) is also placed in the TB.

5. Having placed the system page table entry in the TB, the reference is retried for a guaranteed TB hit. The sequence described for a TB hit to system virtual space is then performed (Figure 1-12, Part A).

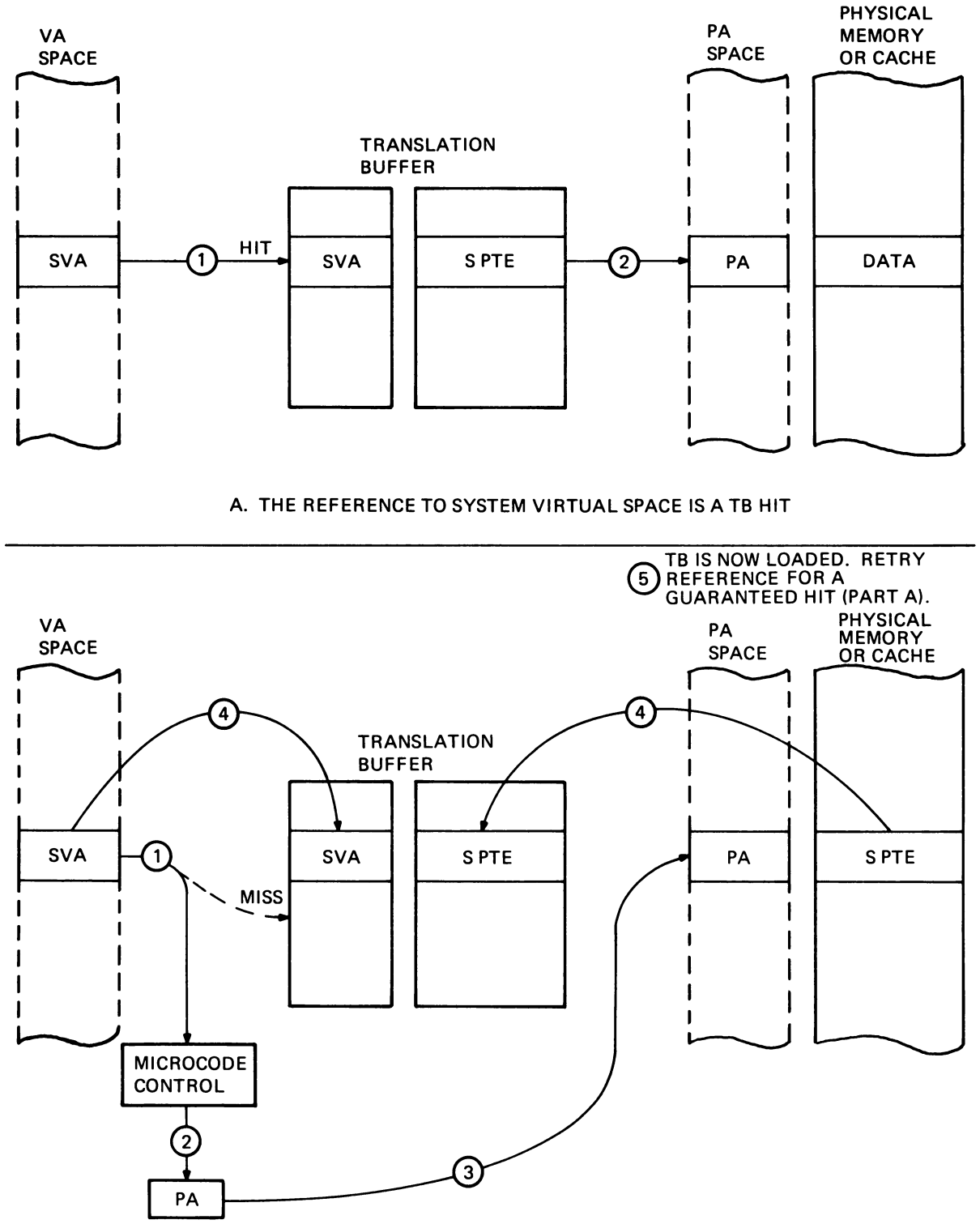


Figure 1-12 Sequence for Reference to System Virtual Space

1.7.1.3 Reference to Process Virtual Space with a TB Hit (Figure 1-13, Part A)

1. The CPU data path generates a process virtual address and sends it to the TB. The address matches a virtual address in the TB (TB hit) which indicates the corresponding process page table entry (PX PTE, where X = 1 or 0 indicating control or program) is in the TB.

2. The page table entry contains the physical address of the desired data. If the physical address is in Cache, the corresponding cache data may be accessed. (A write to Cache results in the simultaneous update of main memory.) If the physical address is not in Cache, main memory must be accessed for the data.

1.7.1.4 Reference to Process Virtual Space with a Single TB Miss (Figure 1-13, Part B)

1. The CPU data path generates a process virtual address and sends it to the TB. The address does not match a virtual address in the TB (TB miss) which indicates the corresponding process page table entry (PX PTE, where X = 1 or 0 indicating control or program) is not in the TB.

2. The CPU microcode calculates the system virtual address of the process page table entry. If in the TB, the corresponding system page table entry will contain the physical address of the process page table entry. (This is because all process page tables exist in system space, and all system space is mapped by the system page table. See Paragraph 1.5).

3. The microcode-generated system virtual address is sent to the TB and a TB hit occurs. (A TB miss at this point is examined in the following sequence example.)

4. The corresponding system page table entry contains the physical address of the process page table entry.

5. The process page table entry is fetched from main memory (or Cache) and placed in the TB along with its corresponding address information (provided the V bit is set).

6. Having placed the process page table entry in the TB, the original reference is retried for a guaranteed TB hit. The sequence described for a TB hit to process virtual space is then performed (Figure 1-13, Part A).

1.7.1.5 Reference to Process Virtual Space with a TB Double Miss (Figure 1-13, Part C)

1. The CPU data path generates a process virtual address and sends it to the TB. The address does not match a virtual address in the TB (TB miss), which indicates the corresponding process page table entry (PX PTE, where X = 1 or 0 indicating control or program) is not in the TB.

2. The microcode calculates the system virtual address of the process page table entry. If in the TB, the corresponding system page table entry will contain the physical address of the process page table entry. (This is because all process page tables exist in system space, and all system space is mapped by the system page table. See Paragraph 1.5).

3. The microcode-generated system virtual address is sent to the TB and another TB miss occurs.

4. Under microcode control, the physical address of the system page table entry is calculated from its virtual address.

5. The physical address locates the system page table entry in main memory (or Cache).

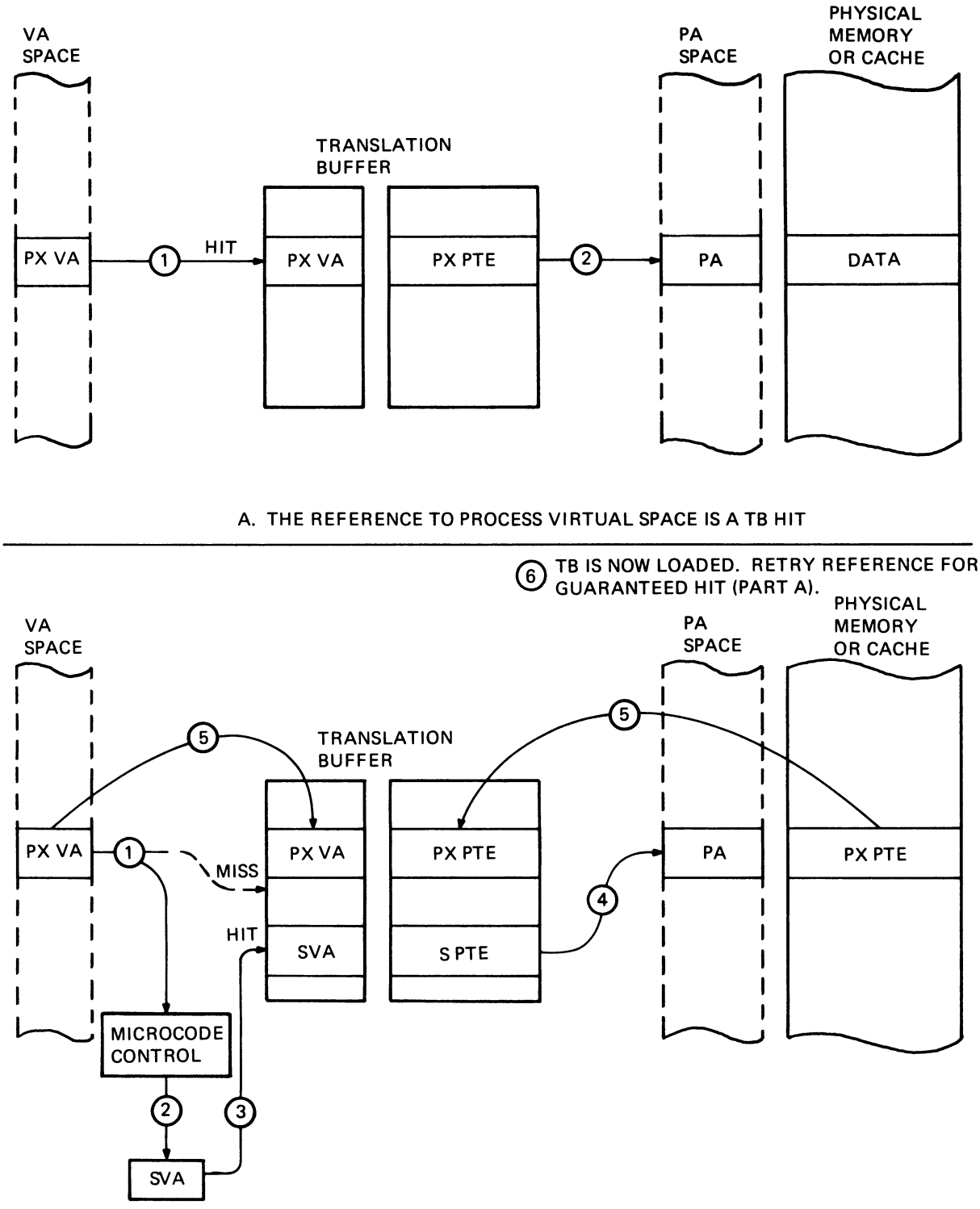


Figure 1-13 Sequence for Reference to Process Virtual Space (Sheet 1)

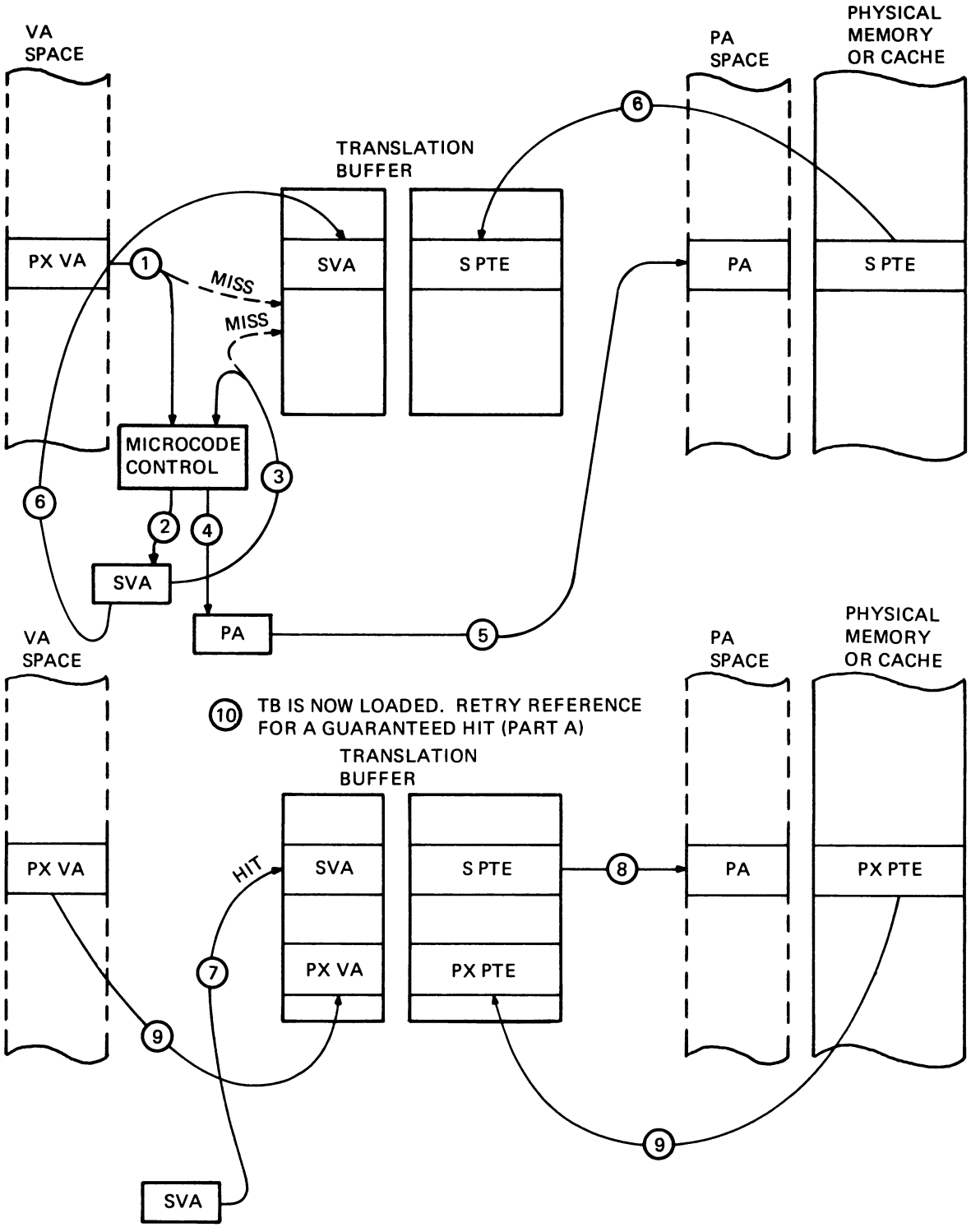


Figure 1-13 Sequence for Reference to Process Virtual Space (Sheet 2)

6. The system page table entry is retrieved from main memory and placed in the TB along with its addressing information.

7. Having placed the system page table entry in the TB, the system virtual reference is retried for a guaranteed TB hit. (The system virtual reference was previously generated by the microcode.)

8. The system page table entry contains the physical address of the process page table entry. Thus the process page table entry is fetched from memory (or Cache).

9. The process page table entry is placed in the TB with its address information.

10. Having placed the process page table entry in the TB, the original reference is retried for a guaranteed TB hit. The sequence described for a TB hit to process virtual space is then performed (Figure 1-13, Part A).

1.8 BUS SUMMARY

As shown in Figure 1-1, the major buses interconnecting the Translation Buffer, Cache, SBI Control, and the remaining portion of the CPU are:

Synchronous Backplane Interconnect (SBI)
Physical Address Bus (PA Bus)
Control Store Bus (CS Bus)
Internal Data Bus (ID Bus)
Memory Data Bus (MD Bus)
Visibility Bus (V Bus).

These buses are briefly described next and further described in the Functional/Logic Description.

1.8.1 Synchronous Backplane Interconnect

The Synchronous Backplane Interconnect (SBI) is the bidirectional information path and communication protocol for data exchanges between the CPU, memory, and adapters of the VAX-11/780 system. The SBI provides checked, parallel information exchanges synchronous with a common system clock.

The communications protocol allows the information path to be time multiplexed, so that a number of information exchanges may be in progress simultaneously. During each clock period (or cycle), interconnect arbitration, information transfer, and transfer confirmation may occur in parallel.

SBI signals are clocked into data latches. All checking and subsequent decision making is based on these latched signals. Error checking logic detects single failures in the information path. However, multiple SBI system failures are not necessarily detected.

Table 1-2 lists the basic SBI characteristics.

1.8.1.1 SBI Unit Definitions – A nexus, which is any physical connection to the SBI, is capable of performing one or more of the functions listed:

1. *Commander* – A nexus which transmits command and address information.

2. *Responder* – A nexus which recognizes command and address information as directed to it and transmits a response.

3. *Transmitter* – A nexus which drives the information lines.

4. *Receiver* – A nexus which samples and examines the information lines.

Table 1-2 Basic SBI Characteristics

Characteristic	Definition
Number of signal lines	84, including check bits
Data path width	32 bits
Physical address space	2^{30} bytes
Bus cycle time	200 ns
Arbitration logic	Decentralized; simultaneous on each interface
Maximum SBI length	3 m (9.84 ft)
Interface chip	DEC 8646, 4-bit transceiver DEC DC101, priority arbitration chip
Bus Cable Properties	
Characteristic impedance	75 ± 7 Ω
Propagation delay	1.1 ± 0.1 ns/ft
Backplane Properties	
Characteristic impedance	75 Ω
Propagation delay	2.0 ns/ft

As an example, consider a CPU which issues a read-type command. It may be considered one of three nexus types, depending on the point in the information exchange.

When the CPU issues the read command, it is a commander since it is issuing command/address information. At the same time it is a transmitter since it is driving the information lines. When the device (responder) returns the requested data, the CPU is considered a receiver, since it examines the information lines and the data is specifically directed to it. In the strict sense, each nexus is a receiver (i.e., examining information lines) in every SBI cycle.

In the case of a memory read exchange, the memory is the responder since it recognizes and responds to a command/address signal. Also, since it examines the information lines, it is a receiver (along with every other nexus on the SBI). When the memory returns the requested data by driving the information lines, it is a transmitter.

1.8.1.2 SBI Signal Groups - The 84 lines of the SBI are divided into these functional groups:

1. Arbitration
2. Information
3. Confirmation
4. Interrupt
5. Control.

1.8.1.2.1 Arbitration Group – The arbitration group sets nexus priority to access the SBI. It determines which nexus of those requesting access to the SBI in a particular cycle will perform an information transfer in the following cycle.

1.8.1.2.2 Information Group – The information group exchanges command/address, data, and interrupts summary information. Each exchange consists of one to three information transfers.

For write-type commands, the commander uses two or three successive SBI cycles. The number of successive cycles required depends on whether one or two data longwords are to be written in the exchange. In the first case, the commander transmits the command/address in the first cycle, and a data longword in the second cycle. In the second case, the commander transmits the command/address in the first cycle, data longword 0 in the second cycle, and data longword 1 in the third cycle.

Read-type commands are also initiated with a command/address transmitted from the commander. However, since data emanates from the responder, the requested data may be delayed by the characteristic access time of the responder. As in a write exchange, the read data will be transmitted using one or two successive cycles depending on whether one or two data longwords were requested.

An interrupt summary exchange is response to a device-generated interrupt to the CPU. The exchange is initiated with an interrupt summary read transfer from the CPU. The exchange is completed two cycles later with an interrupt summary response transfer containing the interrupt information.

1.8.1.2.3 Confirmation Group – The confirmation group provides a path to inform the transmitter whether the information transfer was correctly received and, in the case of a command/address transfer, whether the receiver can process the command.

Each command/address or information transfer is confirmed by the responder (or receiver) two cycles after transmission by the commander. During a write-type exchange, command/address and data transfers are confirmed by the responder. During a read-type exchange, the command/address transfer is confirmed by the responder; the reception of read data is confirmed by the commander.

Interrupt summary transfers are not confirmed.

1.8.1.2.4 Interrupt Request Group – The interrupt request group provides a path for nexus to interrupt the CPU to service a condition requiring processor intervention. In addition, the group includes a special line for nexus which interrupts the CPU only for changes in power or operating conditions.

1.8.1.2.5 Control Group – The control group provides a path to synchronize system activity and provides specialized system communication. The group includes the system clock which provides the universal time base for any nexus connected to the SBI. The group also provides initialization, power fail, and restart functions for the system. In addition, an interlock line is provided for coordination of memory sharing in multiprocessor systems.

1.8.2 Physical Address Bus

The physical address (PA) bus is a bidirectional internal bus 28 bits wide [PA (29:02)]. The PA bus transfers the translated physical address from the TB to the Cache and SBI Control. In the case when the memory management enable function is off, the address transferred is not translated. The PA bus is also used to transfer a physical address from the SBI Control to Cache for Cache refill and SBI invalidated sequences.

1.8.3 Control Store Bus

The control store (CS) bus is a 96-bit wide control bus which is essentially the output of the control store microword.

The Translation Buffer receives control from a microcode field consisting of six lines. Although the SBI Control is not connected to this microcode field directly, the field is buffered and received from receivers in the TB.

1.8.4 Internal Data Bus

The internal data (ID) bus is a high speed, bidirectional data path connection between the TB, SBI control, and other functional areas of the CPU. The ID bus consists of 32 data lines, 6 address lines, and 1 direction line. ID bus control is derived from a control field in the microword during normal operation and from the console interface logic during maintenance operation. The ID bus is further discussed in the *VAX-11 KA780 Central Processor Description* (EK-KA780-TD-PRE).

1.8.5 Memory Data Bus

The memory data (MD) bus is the bidirectional information path for longword aligned data exchanges which connects the data path portion of the CPU and the instruction buffer to the Cache and SBI Control (Figure 1-14). The bus consists of 40 lines: 32 data lines, 4 parity lines, and 4 mask lines. The parity lines provide parity for each of the four data bytes (i.e., parity bit 0 associated with byte 0, bits 7-0, etc.). The mask bits are associated with the data bytes similar to the parity bits. The mask bits inform the system which bytes are to be read or written.

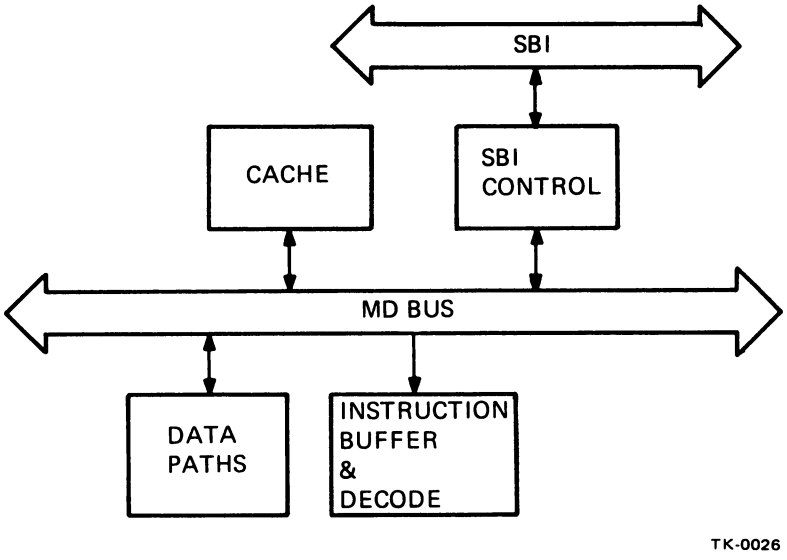


Figure 1-14 MD Bus

The MD bus transfers data for these general cases:

1. Data path requested read data is found in Cache (hit) and transferred back to the data path via the MD bus.

2. Instruction buffer read data is found in Cache (hit) and transferred back to the instruction buffer via the MD bus.

3. Data path requested read data is not found in Cache (miss) and is retrieved from main memory. The data is transferred from the SBI Control to the data path and Cache simultaneously via the MD bus.

4. Data requested by the instruction buffer is not found in Cache (miss) and is retrieved from main memory. The data is transferred from the SBI Control to the instruction buffer and Cache simultaneously via the MD bus.

5. CPU write data is transferred to the SBI control via the MD bus and sent over the SBI to be written in memory. If it is in Cache, the data is also updated in Cache simultaneously via the MD bus.

6. Interrupt Summary Read Responses are transferred over the MD bus to the data path.

1.8.6 Visibility Bus

Various signals from the TB, Cache, and SBI Control are interfaced to the V bus for diagnostic isolation of CPU subsystem failures. Refer to the Console Technical Description (EK-KC780-TD-PRE) for a detailed explanation of the V bus and its operation.

1.9 TRANSLATION BUFFER OVERVIEW

The TB is a two-way set associative cache used to store page table entries. The TB is constructed of two major parts: TB data matrix and TB address matrix, where each matrix consists of 128 entries. The components of the TB, in conjunction with the firmware translation routines, perform the virtual-to-physical address translation.

1.9.1 TB Structure

As shown in Figure 1-15, the TB address and data matrices are each divided into two groups. Since each group contains 64 locations, the TB has a total capacity of 128 locations. Each entry of the group 0 address matrix corresponds to an entry in the group 0 data matrix. Likewise, each entry of the group 1 address matrix corresponds to an entry in the group 1 data matrix.

1.9.2 TB Functions

Portions of each PTE in the TB are stored in the address matrix and portions are stored in the data matrix. The address matrix stores addressing information (virtual tag field) and protection information for the corresponding pages. The data matrix contains the translated physical page frame number (i.e, the high-order address bits) of each PTE.

As shown in Figure 1-15, the virtual address (VA) is divided into three fields. The index field selects a location in both groups of the address and data matrices. Once selected, the tag from both groups of the address matrix is compared against the tag of the virtual address.

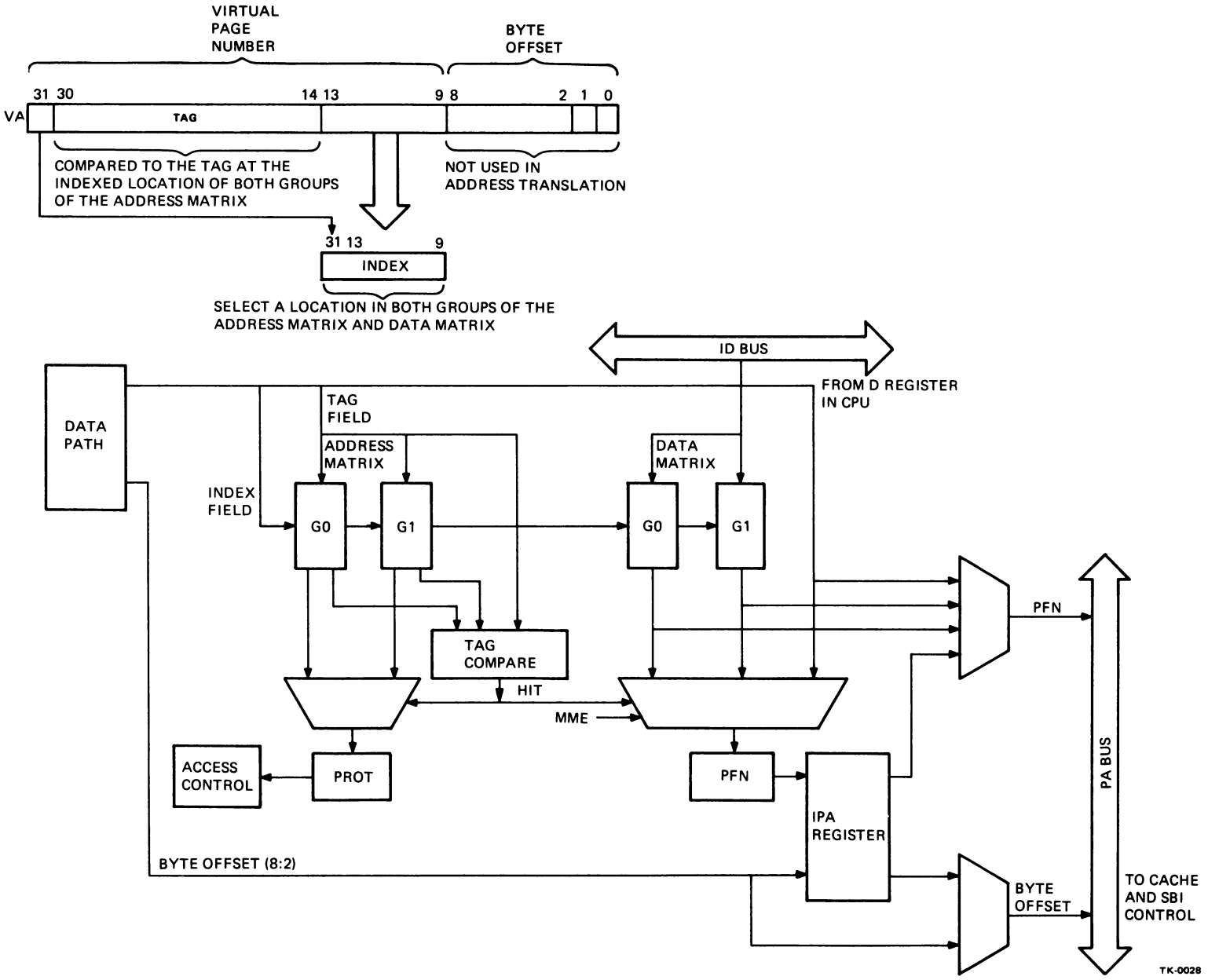


Figure 1-15 Basic Translation Buffer Structure

If a match is found between either group tag and the VA tag field, the reference is considered a TB hit and portions of the associated PTE are retrieved from the corresponding group of the address and data matrices. The PFN of the page table entry is enabled to the PA bus, combined with the byte offset from the virtual address, and transferred to Cache as the physical address for a data word lookup. The protection part of the PTE is checked for possible access violation. If there is no match between the TB tags of the indexed location and the VA tag, the reference is considered a TB miss and a microtrap occurs. During the microtrap, the microcode fetches the PTE from the page table in memory and writes it in the TB via the ID bus. The reference is then retried. If the V bit of the PTE within the page table is not set, a trap to system software occurs. (System software then locates the page, updates memory and the PTE, and retries the reference.)

1.9.3 IPA Operation (Instruction Physical Address)

Whenever any type of branch is executed, the microcode must calculate the new address, translate it, and load it in the IPA register. The IPA register is then incremented to generate consecutive addresses until another branch is executed.

When the IPA register is incremented across a page boundary, the address of the new page must be translated and checked for accessibility. An auto-reload feature in the hardware automatically translates the new address and loads it in the IPA without the need of microcode control.

1.10 CACHE OVERVIEW

The Cache is two-way set associative and is used to store data quadwords retrieved from memory via the SBI Control during processing to speed system operations. As shown in Figure 1-16, the Cache has two major parts: cache data matrix and cache address matrix. Each matrix is divided into two groups (Group 0 and Group 1), with the matrix groups having the same general relationships as in the TB. The address matrix stores the tag field (high-order physical address bits) of one associated data quadword stored in the data matrix. Each address matrix entry corresponds to one data matrix entry consisting of two longwords.

The Cache address matrix has two groups with 512 entries in each group. Correspondingly, the Cache data matrix consists of two groups with 512 entries in each group. Since each data entry consists of two longwords, the total data capacity of Cache is 2048 longwords.

1.10.1 Cache Function

The physical address presented to cache is divided into the fields shown in Figure 1-16. The index fields are used to retrieve the referenced index position in both groups of the address and data matrices. However, the index fields are of different lengths. A 9-bit index field [PA (11:3)] is required to reference the 512 locations of the address matrix. The data matrix requires a 10-bit index field [PA (11:2)], since a data entry has two longwords. PA bit 2 selects the longword of the quadword entry.

During a Cache operation, the tag fields (retrieved from Group 0 and 1 of the address matrix) are compared with the tag field of the physical address from the PA bus. If a match is found between either group tag and the physical address tag field, the reference is considered a hit. In a read operation the data longword stored in the associated group of the data matrix is enabled to the instruction buffer or D register in the data path via the MD bus. In a write operation, a hit updates the indexed data location.

NOTE

The location in main memory is updated on a write, whether or not a cache hit occurs.

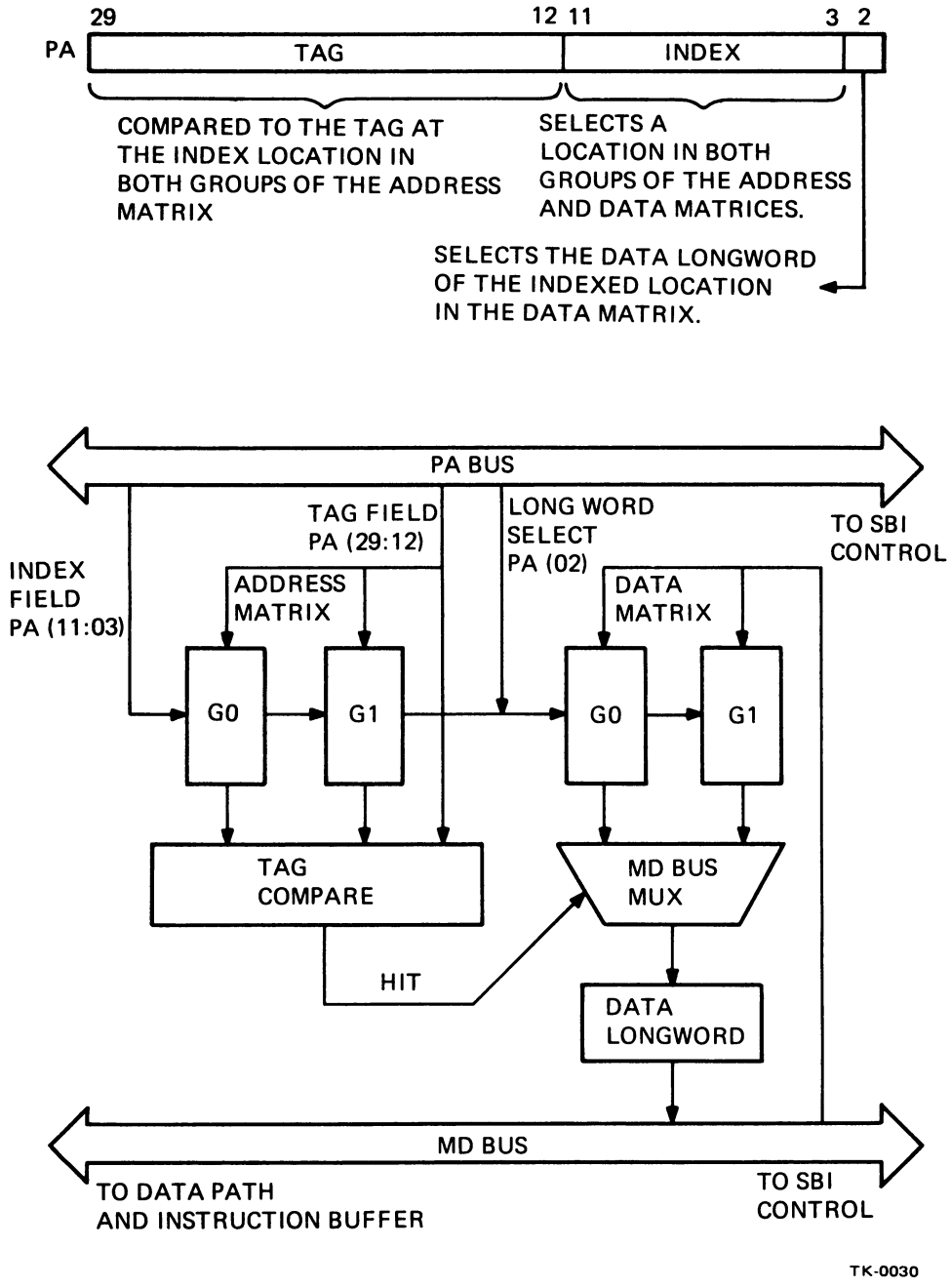


Figure 1-16 Basic Cache Structure

In the case of a read miss (reference not in Cache), the entire quadword containing that location is retrieved from main memory (an Extended Read operation). Both longwords are written into the cache, and the referenced data is sent to the instruction buffer or D register in the data path. In the case of a write miss, the referenced location is updated in main memory only (Paragraph 1.10.2.).

1.10.2 Cache Strategies

The cache uses a random replacement strategy. That is, when new data is written in the cache from main memory, the group used is chosen in a pseudo random manner rather than overwriting on a least recently used or first-in, first-out basis. A flip-flop is used as the random bit which complements every cycle until a cache miss occurs.

The Cache uses a modified write-through updating strategy. When the CPU does a write cycle, that location is updated in Cache (if in Cache) and also updated in main memory. However, since the SBI Control can buffer one command, the CPU is not forced to wait for the write cycle to complete before continuing processing (assuming no error conditions). The CPU is forced to wait in the case of two successive write cycles or an instruction buffer read miss followed by a write (Paragraph 1.12.2).

The miss strategy implemented in the Cache is not write allocate. In the case where the CPU does a write cycle and has a write miss (reference not in Cache), that location is updated in memory. The location, however, is not stored in Cache.

1.11 SBI CONTROL OVERVIEW

The major function of the SBI Control is to transfer data between the CPU and other system components on the SBI. It has the capability to initiate interrupts and microtraps as determined by conditions on the SBI or in other areas of the CPU (Figure 1-17.)

The SBI control logic is implemented on two extended hex-height boards. In addition to the associated control logic, the 32-bit data path is evenly divided between the two boards (16 bits/board).

1.11.1 Basic Operations

The SBI Control firmware commands are buffered in the TB. On a read operation having a Cache read miss, the SBI Control initiates an SBI cycle to retrieve the quadword containing the referenced location from main memory. Having received the address simultaneously with Cache over the PA bus, the address may be enabled from the PA register to the SBI. When the quadword is retrieved, the SBI Control assumes control of the PA and MD buses. The contents of the PA register are then enabled back to the PA bus and the retrieved data quadword is transferred to Cache to be written in the indexed locations of the randomly selected group.

On a write operation the address and write data are transferred to buffers in the SBI Control. As soon as the SBI Control gains control of the SBI, the write data is transferred to main memory. (The write data buffer allows the CP to resume processing without waiting for the SBI write cycle to complete.)

1.12 COMBINED OPERATIONAL OVERVIEW

The following subsections provide an overview of the combined basic operations of the Translation buffer, Cache, and SBI Control. Refer to Figure 1-18.

1.12.1 Basic Read Operation

Initially, the CP data path section transfers a virtual address (VA) generated during process execution, over the VA lines to the TB together with a firmware command from the control store. Note that the firmware command is simultaneously applied to the Cache and SBI Control. In this case the firmware command directs the subsystem to execute a virtual read operation.

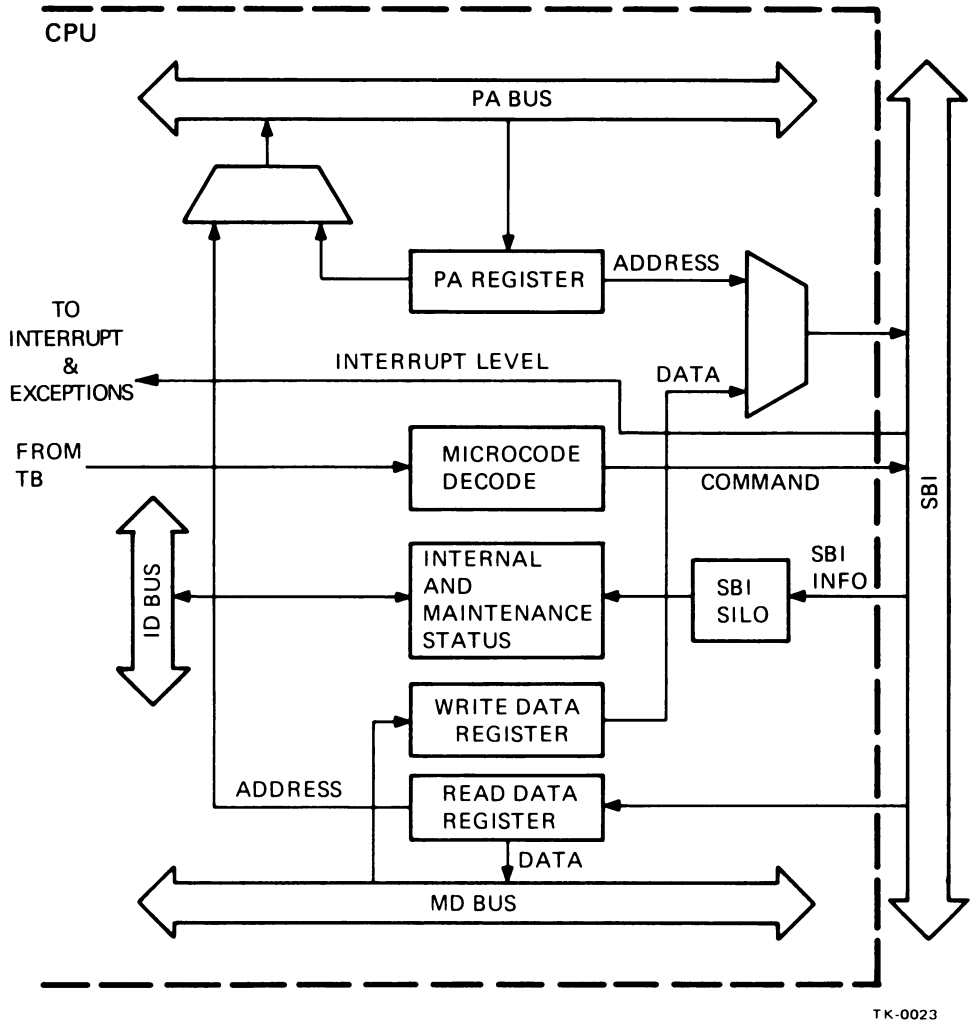


Figure 1-17 Basic SBI Control Structure

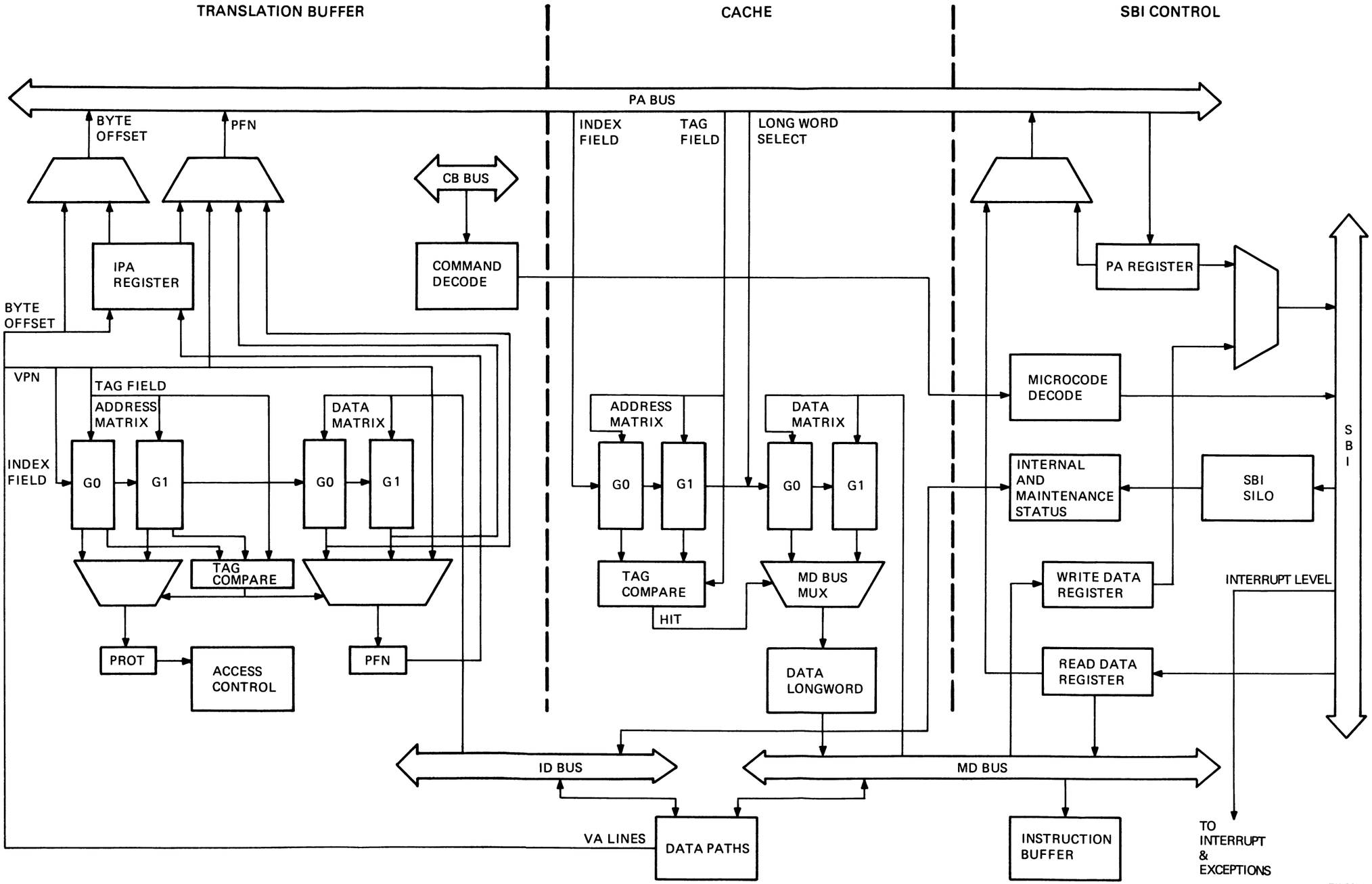


Figure 1-18 TB, Cache, SBI Control Basic Block Diagram

The VA tag and index fields are applied to both groups of the TB address and data matrices (Figure 1-18). The tag fields retrieved from Groups 0 and 1 of the address matrix are compared with the tag field of the VA presented to the TB for translation.

If there is no match between either group tag field and the VA tag field (TB miss), the microcode retrieves the associated PTE from memory, places it in the TB, and then causes the reference to be retried.

If a match is found (TB hit) between either group tag field and the VA tag field, the associated PTE information (translated page frame number) in the TB data matrix is enabled to the PA bus. Since the index fields are simultaneously applied to the TB address and data matrices, when a hit occurs in the address matrices, the corresponding contents of the data matrix is available for transfer at hit time. The page frame number and byte offset are then transferred over the PA bus to Cache for a data word lookup and the SBI Control, in case a Cache miss occurs.

The cache control logic initiates a lookup to determine if the referenced data is stored in Cache. The physical address from the PA bus is applied to both groups of the cache address and data matrices as shown in Figure 1-18. The index fields are used to retrieve the referenced index position in both groups of the address and data matrices. The tag fields retrieved from Groups 0 and 1 of the address matrix are compared with the tag field of the physical address from the TB. If a match is found (Cache hit) between either group tag field and the PA tag field, the referenced data word is in the associated group of the data matrix.

As in the TB, the index fields are simultaneously applied to the cache address and data matrices. Thus, the associated data matrix content is available for transfer at hit time. If the referenced data is in the matrix, it is enabled to the MD bus and transferred back to the D register in the data path or the instruction buffer.

If the data is not present in Cache, the cache control logic notifies the SBI control that the data is not available. Since the PA and firmware command were applied to the SBI Control at the same time they were applied to Cache, the SBI Control initiates an SBI read cycle to retrieve the referenced data from main memory. CPU normal execution is suspended until this data is returned from main memory. When the data is returned from main memory, the SBI Control assumes control of both the PA and MD buses. The PA is enabled back to the PA bus and the retrieved data is transferred to the data path or instruction buffer and also written into the indexed location of a randomly selected group in the cache data matrix.

Note that the SBI read cycle to main memory retrieves two longwords (quadword) rather than only the longword that was requested (an extended read operation.) This is in anticipation that the CPU will reference the next sequential longword during the current processing.

1.12.2 Basic Write Operation

As in the read operation, the CP transfers a VA to the TB and a firmware command to the TB, Cache, and SBI Control. Following address translation, the PA is transferred over the PA bus to the Cache and SBI Control. The Cache then performs a lookup in the address matrix for the referenced data (tag field compare).

If the Cache data matrix contains the referenced address, the content of that location is updated in the Cache and an SBI write cycle is initiated to update the referenced location in main memory. If the reference is not in the address matrix, no write to the data matrix will be executed. Having latched the address from the PA bus simultaneously with Cache, the SBI Control initiates an SBI write cycle to update the referenced location in main memory only. Note that, in keeping with the not write allocate cache strategy, the missed location is not brought to the data matrix.

The data buffering in the SBI Control improves CPU performance during writes. The CP is not forced to wait for the SBI write cycle to complete before proceeding with its normal processing. However, the CPU is forced to wait during:

1. Two successive write cycles or

2. An instruction buffer read miss followed by a write.

For two successive write cycles, the PA register and Write Data register of the SBI Control initially became filled with the address and data of the first write. In this case the second write must wait until these registers are emptied onto the SBI. For the case of an instruction buffer read miss followed by a write, the PA register must not only hold the address for transmission over the SBI, but also must retain the address until the requested read data is latched from the SBI. When the read data is received, the PA register places the address on the PA bus for a Cache update. With this, the PA register becomes available for the write address. Thus, when an instruction buffer read miss is followed by a write, the write must wait until the read data is retrieved.

1.13 MODULE LOCATIONS

The six extended hex-height boards of the TB, Cache, and SBI Control are listed in Table 1-3. The table also includes their slot location in the KA780 backplane.

Table 1-3 TB, Cache, and SBI Control Modules

Module Type		Module Title	Slot Location
M8222	TBM	Translation Buffer Matrix	6
M8221	CDM	Cache Data Matrix	5
M8220	CAM	Cache Address Matrix	4
M8219	SBH	SBI High Bit Interface	3
M8218	SBL	SBI Low Bit Interface	2
M8237	TRS	SBI Terminator plus Silo	1

CHAPTER 2
FUNCTIONAL/LOGIC DESCRIPTION

The organization of the Function/Logic Description is similar to that of the Overview. The TB structure and logic is discussed first with a more detailed description of the address translation process. General Cache concepts are also provided as an introduction to the discussion of the Cache of the VAX-11/780. The Cache discussion is followed by a description of the SBI Control (including SBI protocol) and its various ID bus registers. Finally, the microcode initiated memory control functions are discussed as a summary of the overall operation.

2.1 TRANSLATION BUFFER DESCRIPTION

2.1.1 Translation Buffer Matrix Structures

As mentioned previously, the Translation Buffer is actually a cache of page table entries. In the translation buffer, a virtual address (consisting of index and tag fields) selects a PTE. The PTE contains the upper 21 bits of a physical address and protection information. The organizations of the address and data matrices are illustrated in Figure 2-1.

As seen in Figure 2-1, portions of each PTE are stored in the address matrix and portions are stored in the data matrix. The V bit, M bit, and protect field are located in the address matrix and the 21 physical address bits are located in the data matrix. Each address matrix and data matrix entry also contains parity bits.

Each address and data matrix is divided into two identical groups, Group 0 and 1. Each group contains 64 locations. Every location in the address matrix corresponds to a location in the data matrix. Half of the 64 locations are reserved for system PTEs and half are reserved for process PTEs. This organization makes it possible to clear all process PTEs in the buffer for a context switch without clearing system PTEs. With this, recalculations of system PTEs are not required for every context switch.

2.1.2 TB Operation - General

The address translation algorithms are actually executed by the CPU microcode. The TB merely stores the results of the translation algorithm for reuse, thereby saving time when the address is needed again. Figures 2-2 and 2-4 contain flow diagrams of address translation. Figure 2-2 illustrates the translation of a reference to system virtual space. Figure 2-4 illustrates the translation of a reference to process virtual space. As seen in these figures, a translation begins when the data path presents the TB with a virtual address. If the TB contains a tag identical to that of the incoming address (and the valid bit is set), the reference is a hit which indicates the TB also contains the page table entry. If the TB does not contain an identical tag, the reference is a miss and a microtrap occurs.

Although only valid PTEs are loaded into the TB, the TB may contain invalid PTE's due to invalidation by the operating system. A TB entry is invalidated by the operating system when the corresponding page is removed from memory and placed on disk (i.e., removed from the working set).

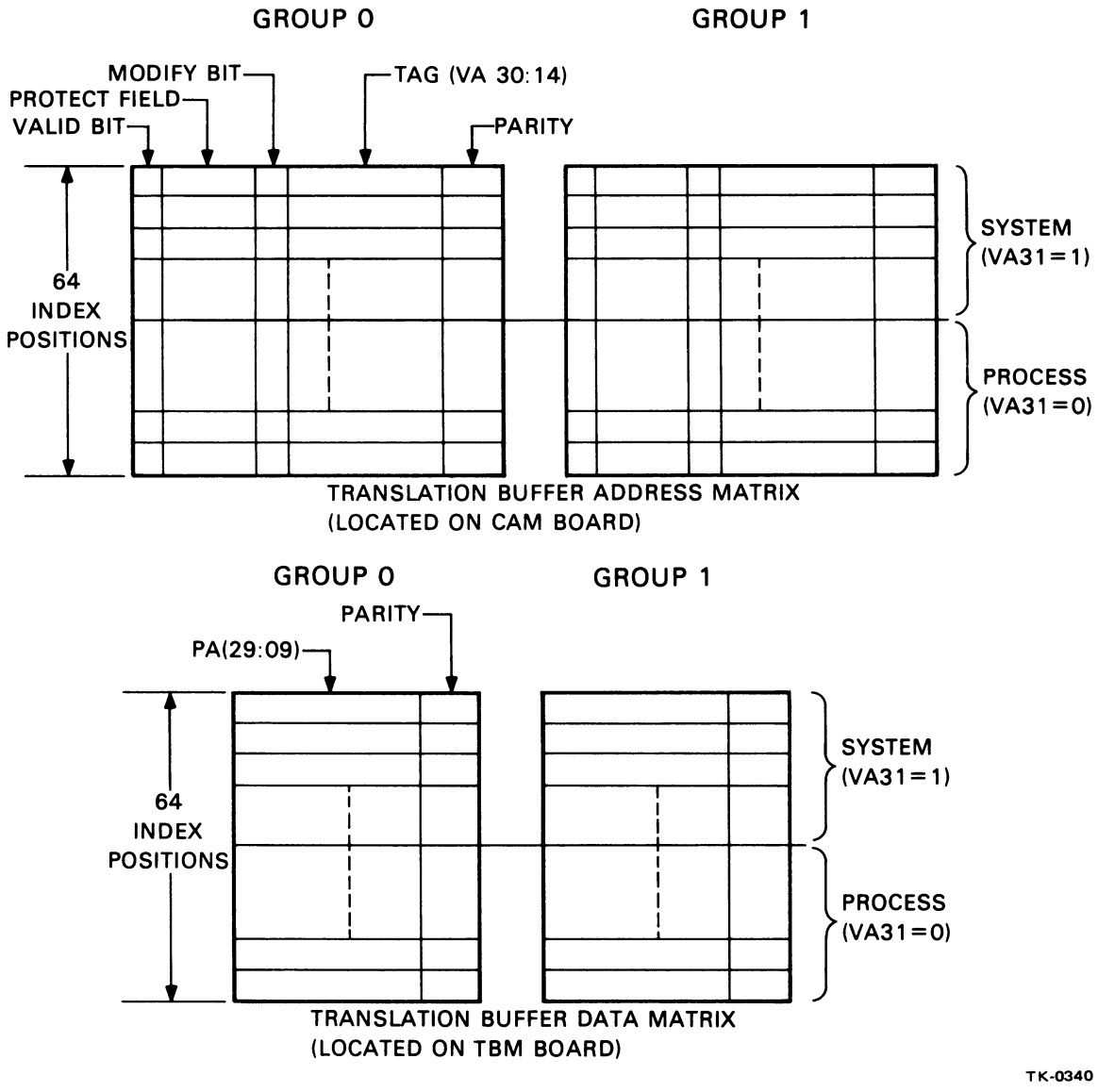


Figure 2-1 Translation Buffer Matrix Structures

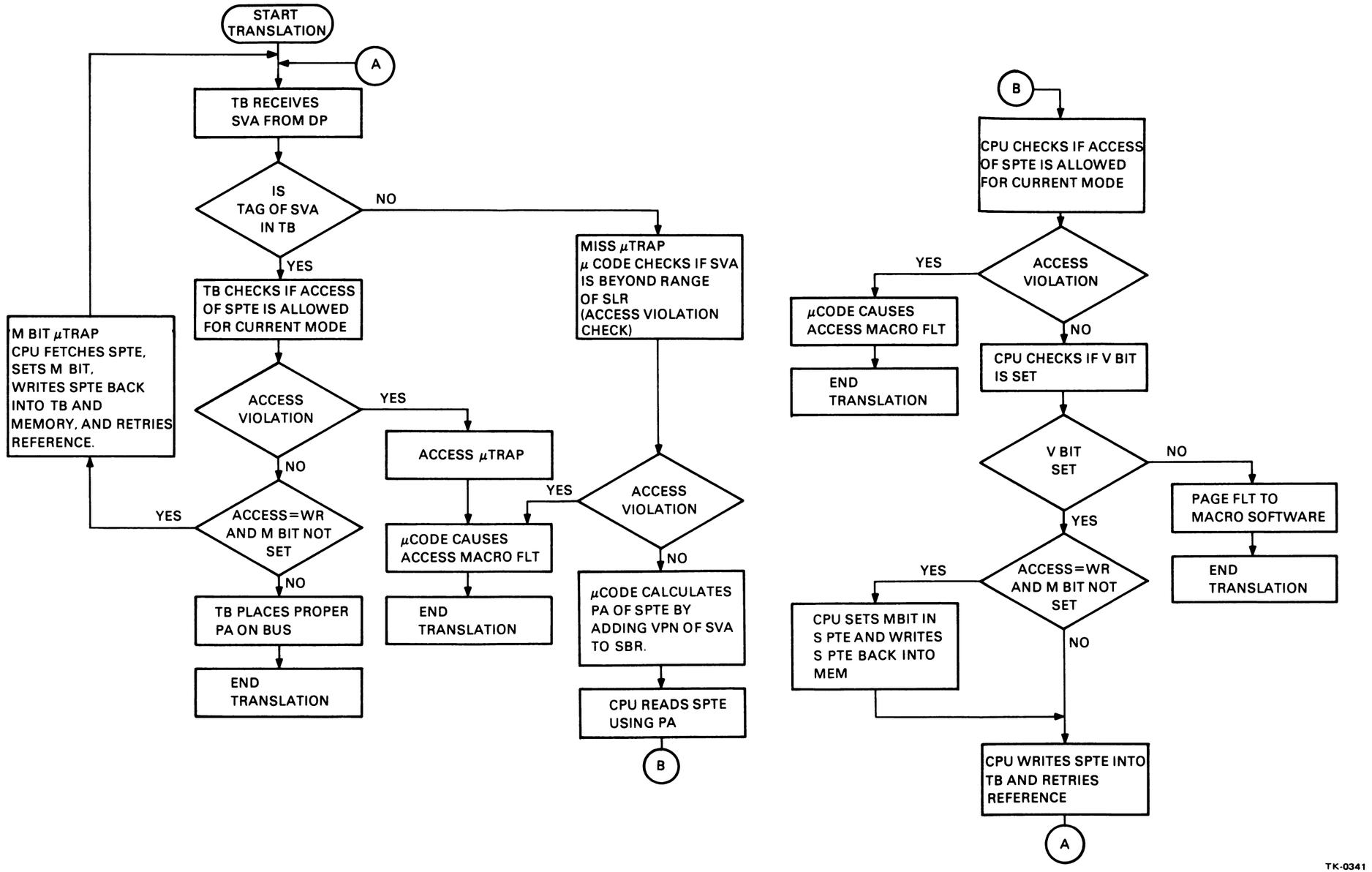


Figure 2-2 Translation of Reference to System Virtual Space

2.1.2.1 System Virtual Reference, TB Hit – For references to system space, bit 31 of the incoming address equals 1. As seen in the flow diagram in Figure 2-2, if the TB contains the tag, the reference is a hit (TB contains the SPTE) and the protect field of the SPTE is compared against the current mode of the CPU (Paragraph 2.1.3). An access violation in this case causes a microtrap and a macrofault which aborts the translation. If there is no access violation, an M bit check is executed.

An M bit microtrap occurs if the access is a write and the M bit is not set. This indicates the write is the first modification to the page. The M bit microtrap fetches the SPTE from main memory, sets the M bit, and then rewrites the SPTE back to main memory and the TB. (This is done to notify the operating system that the page has been modified while in main memory and must be rewritten on disk when it exits the working set.) When the M bit microtrap is complete, the virtual address is again sent to the TB to restart the translation.

If an M bit microtrap does not occur, the TB enables multiplexers to output the proper physical address to the PA bus.

2.1.2.2 System Virtual Reference, TB Miss – If the TB does not contain the tag of the incoming address, the reference is a miss (TB does not contain the SPTE) and a microtrap occurs. During the microtrap routine the microcode first executes a page length check which verifies that the system virtual address (SVA) is within the range specified by the system length register (SLR). If it is beyond the page table length, the microcode causes an access macrofault and the translation ends. Otherwise the translation continues under the control of the microcode.

Because the TB does not contain the required SPTE, it must be fetched from main memory (or Cache). The microcode calculates the physical address of the SPTE by adding the virtual page number (VPN) of the SVA to the contents of the System Base register (SBR). Figure 2-3 illustrates this procedure. As seen in this Figure, once the S PTE is retrieved, the byte offset (BO) from the SVA may be added to the page frame number (PFN) to generate the requested physical address.

When the CPU retrieves the S PTE, it is loaded in the TB. The microcode then uses the TB logic to check if the access is allowed for the current processor mode. This is accomplished by using a test operation which checks access but does no memory cycle. If an access violation is detected, however, the microcode only causes a macrofault and the translation ends. Because a test operation is used, there is no microtrap. An access microtrap is not desirable in this case because the violation occurred during a microtrap routine. (A microtrap during a microtrap routine would be difficult for the microcode to handle. Therefore, it is avoided by proper coding.)

With the access of the S PTE verified for the current processor mode, the V bit is checked. The V bit indicates the validity of the S PTE. The V bit is not set if the S PTE is invalid. For this case a page fault to the macro software occurs and the translation ends. If the V bit is set, an M bit check is executed.

An M bit check examines the condition of the M bit. If it is not set and the access is a write, the M bit is set and the S PTE is written back into the TB and main memory. Otherwise the S PTE is only written in the TB. Note that an M bit violation during a miss microtrap does not initiate another microtrap. (A microtrap cannot occur during a microtrap.)

With the correct S PTE in the TB, the address translation is retried. For this, the data path again sends the virtual address to the TB. With the corresponding S PTE just loaded, a TB hit is guaranteed. The sequence described in Paragraph 2.1.2.1 is then followed.

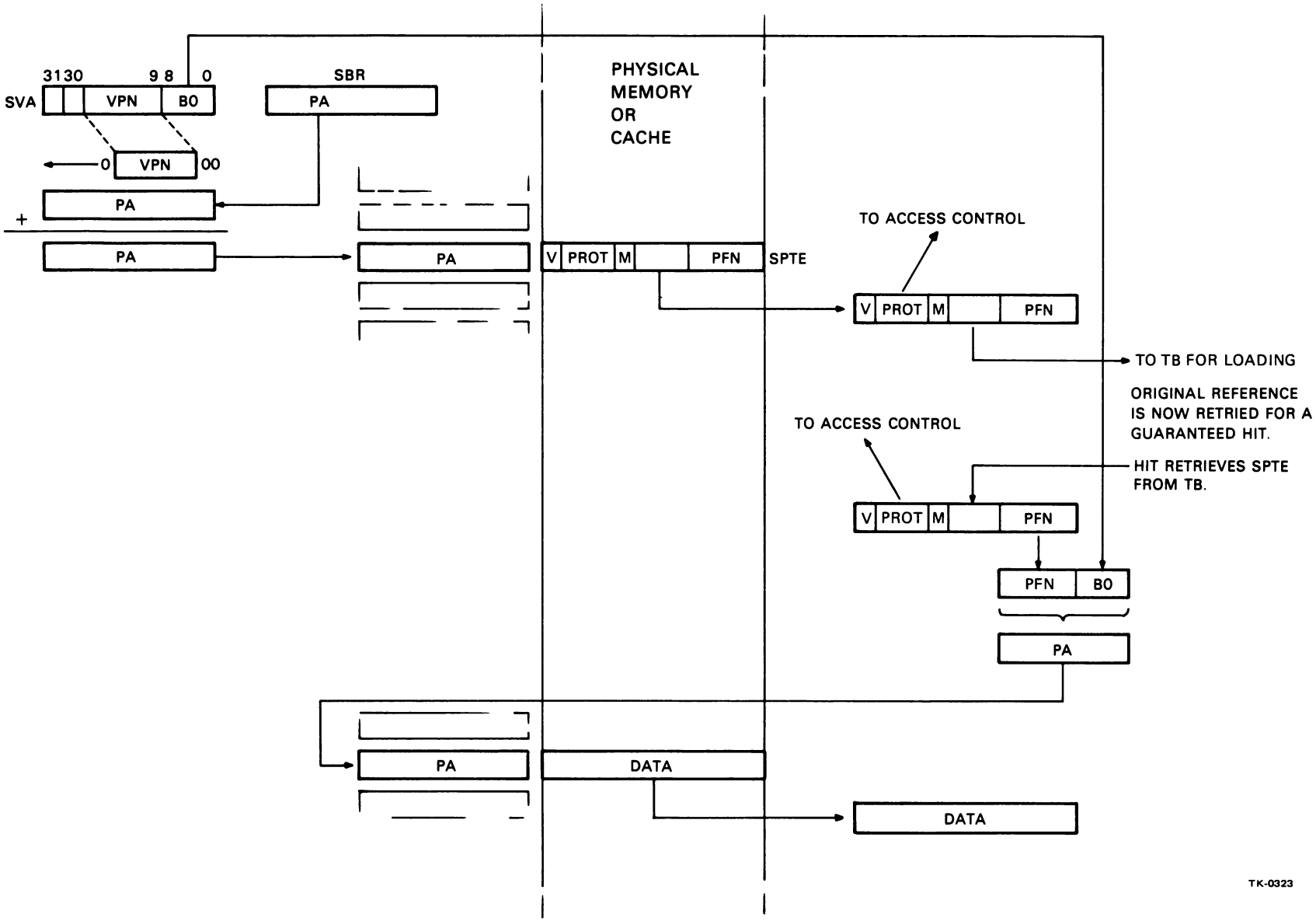


Figure 2-3 Address Calculation for TB Miss
on Reference to System Virtual Space

2.1.2.3 Process Virtual Reference, TB Hit – For references to process virtual space, bit 31 of the incoming address equals 0. As seen in the flow diagram of Figure 2-4, if the TB contains the tag of the incoming address, the reference is a hit (TB contains the PTE) and a procedure similar to the one described for a hit on a system space reference is followed (Paragraph 2.1.2.1).

2.1.2.4 Process Virtual Reference, TB Miss (Single and Double) – If the TB does not contain the tag of the incoming address, the reference is a miss (TB does not contain the PTE) and a microtrap occurs. During the microtrap routine, the microcode first executes a page length check which verifies that the virtual address is within the range specified by the corresponding process length register (P0LR for program space, P1LR for control space). If it is beyond the page table length, the microcode causes an access macrofault and the translation ends. Otherwise the translation continues under the control of the microcode.

Because the TB does not contain the required PTE, it must be fetched from main memory (or Cache). The microcode calculates the system virtual address (SVA) of the process PTE by adding the virtual page number (VPN) of the process virtual address to the contents of the corresponding base register (P0BR for program space, P1BR for control space). This system virtual address is then presented to the TB for translation to a physical address.

As illustrated in Figure 2-4, the procedure described for a reference to system virtual space is followed when the TB is presented with the system virtual address generated by the microcode. However, because this entire procedure occurs during a microtrap, the microcode performs look-ahead tests to eliminate the possibility of a microtrap during a microtrap. If a condition which would normally cause a microtrap should occur, the microcode executes a microbranch. During the microbranch, microcode similar to the microtrap routine is performed.

At the completion of the translation of the system virtual address, the resultant physical address is used to fetch the process PTE from memory (or Cache). When the PTE is retrieved, the TB checks if the access is allowed for the current processor mode. Likewise the V bit check and M bit check are performed just as described for a retrieved system page table entry (Paragraph 2.1.2.3). Assuming no macrofaults occurred, the process PTE is written in the TB end the original reference is retried. With the TB loaded with the correct process PTE, a hit is guaranteed. The procedure for a hit to process space is then performed (Paragraph 2.1.2.3).

2.1.2.4.1 Address Calculation During a Miss Microtrap – During the miss microtrap routine on a reference to process virtual space, the TB is presented with the system virtual address of the process PTE as calculated by the microcode. A TB hit or miss can result. Figure 2-5 illustrates the procedure for a TB hit and Figure 2-6 illustrates the procedure for a TB miss.

As seen in Figure 2-5, the S PTE is read out of the TB when the TB hit occurs. The page frame number (PFN) from the S PTE and byte offset from the SVA are used to generate the physical address of the process PTE. With this physical address, the process PTE is fetched from main memory (or Cache). When retrieved, the PFN of the process PTE and the byte offset of the original process virtual address are used to generate the physical address of the data.

If the system virtual address presented to the TB had resulted in a TB miss, the system PTE would also have had to be fetched from memory (or Cache) just as the process PTE. This is illustrated in Figure 2-6. As seen in this Figure, the VPN of the SVA is added to the contents of the System Base register to yield the physical address of the system PTE. With this physical address the S PTE is fetched from main memory (or Cache). The PFN of the retrieved S PTE is used with the byte offset from the SVA to generate the physical address of the process PTE. The process PTE is then fetched. When retrieved, the PFN of the process PTE and the byte offset of the original process virtual address are used to generate the physical address of the data.

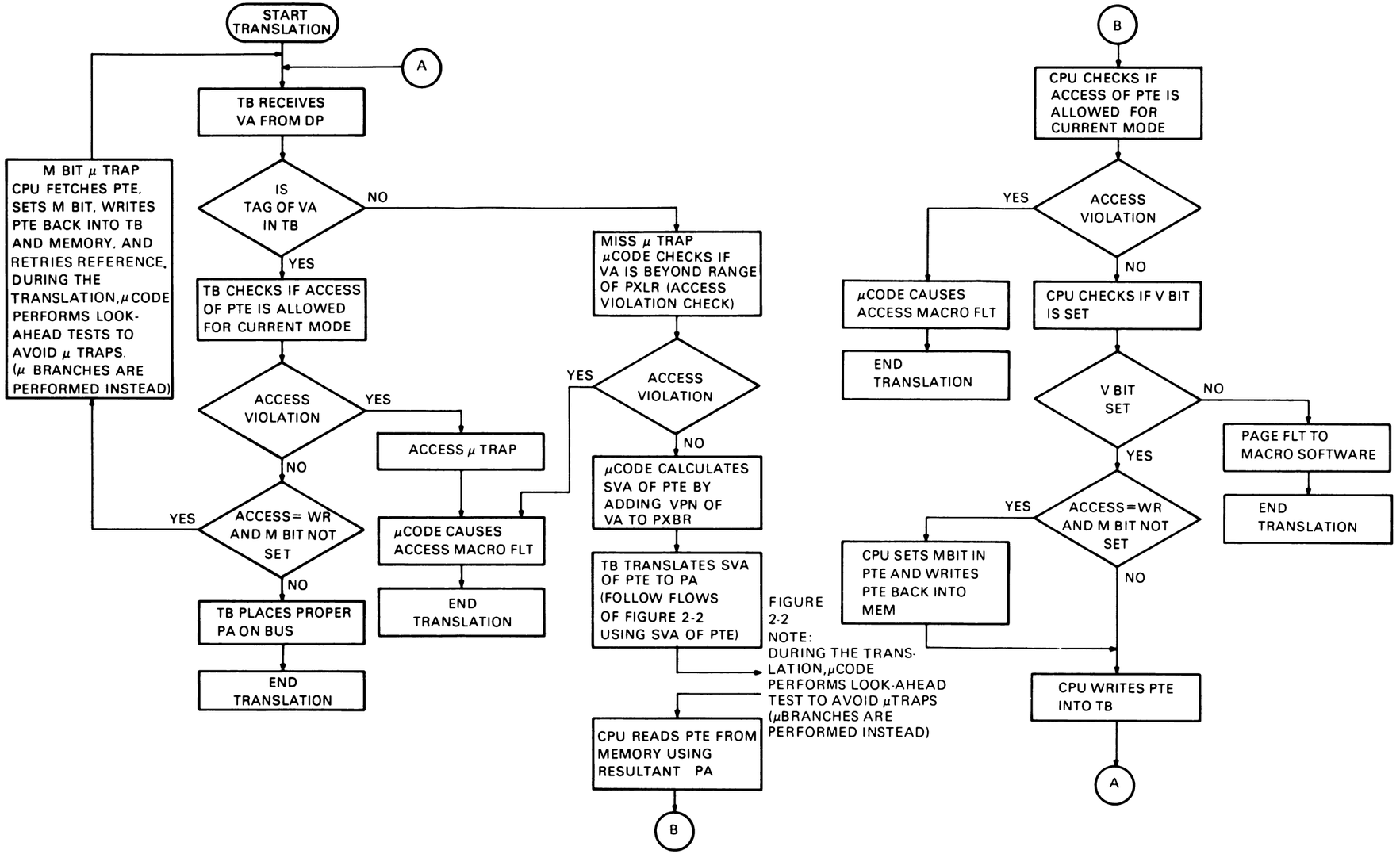


Figure 2-4 Translation of Reference to Process Virtual Space

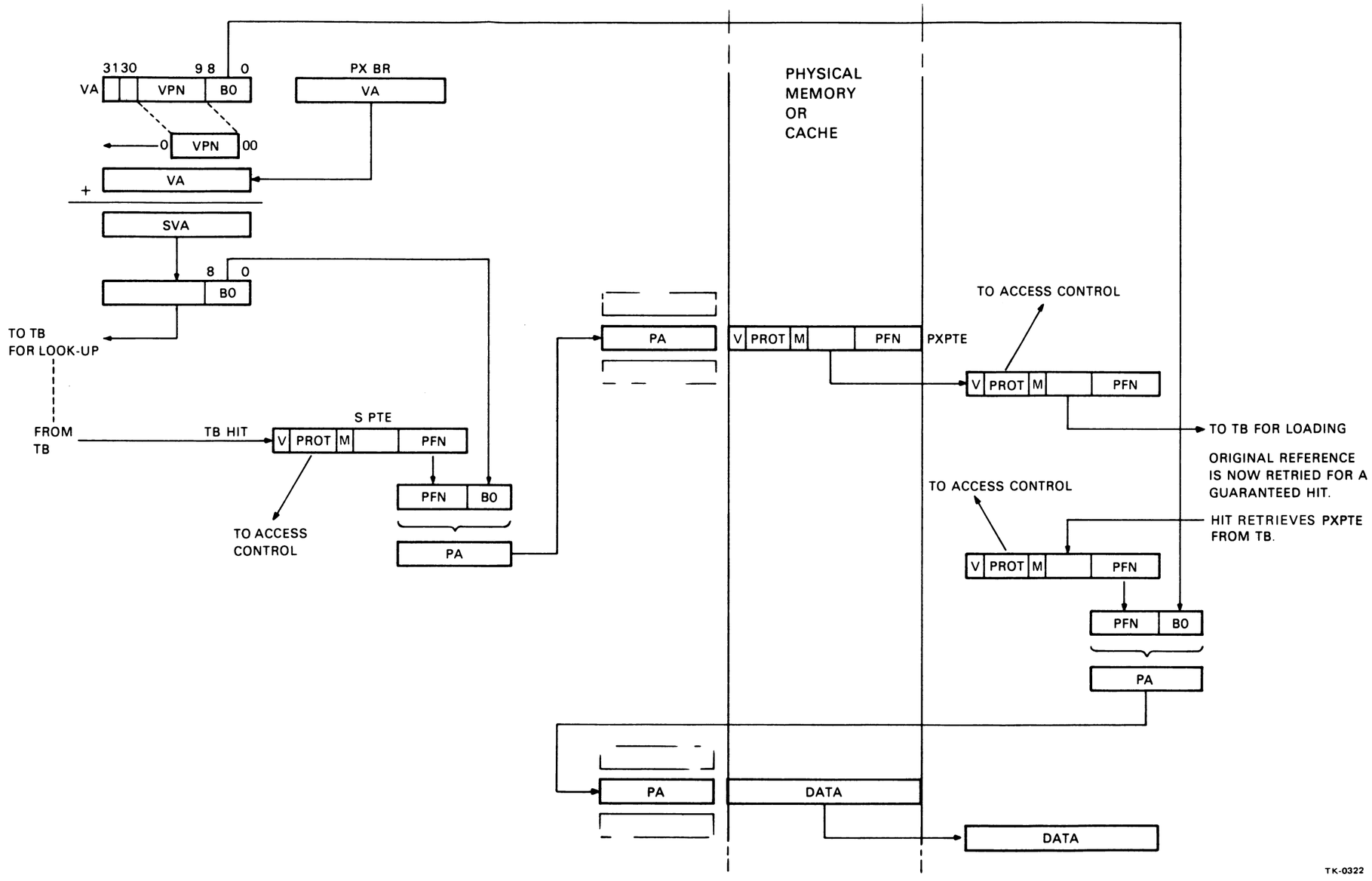


Figure 2-5 Address Calculation for TB Hit During Miss Microtrap

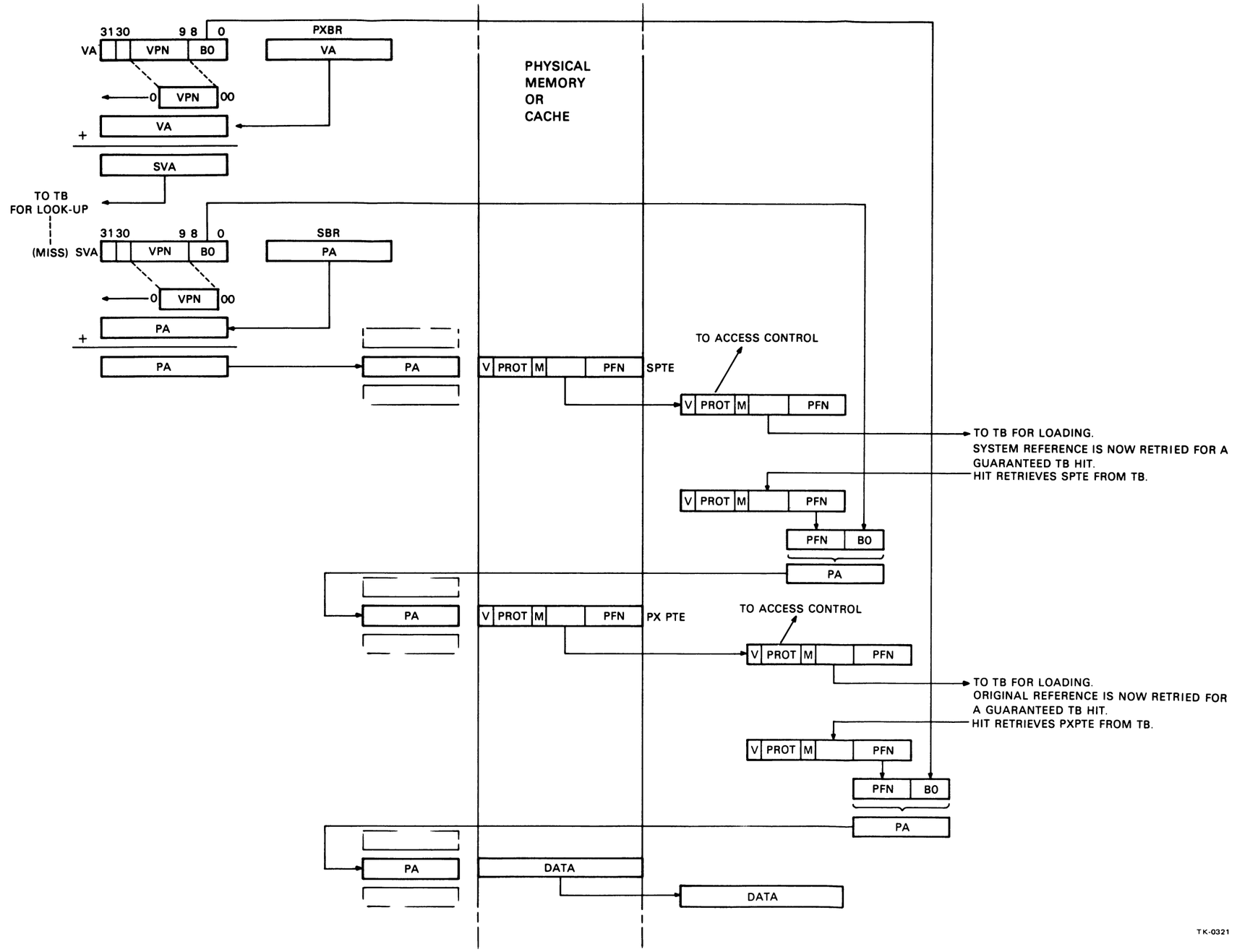


Figure 2-6 Address Calculation for TB Miss During Miss Microtrap

2.1.3 Page Protection

Every PTE contains a protection code for the corresponding page. The protection code of the PTE specifies whether or not read or write references are allowed to the corresponding page. The mode in which the processor is currently running is stored in the current mode field of the processor status longword (PSL). The mode specified for each field content is shown below:

CM Field Content	Mode Specified
00	Kernel
01	Executive
10	Supervisor
11	User

Page accessibility is shown in Table 2-1 for each protection code during each mode. A protection check is executed by the TB whenever a PTE is read from the TB.

2.1.4 IPA Introduction

The Translation Buffer is used to translate addresses from the VA (Virtual Address) register for data references and addresses from the VIBA (Virtual Instruction Buffer Address) register for instruction references. Use of the IPA (Instruction Physical Address) register eliminates the unnecessary translation of each of the consecutive addresses during a string of instructions. The VA register and VIBA register are located in the data path and the IPA register is located in the Translation Buffer (Figure 2-7).

A string of instructions occupies a number of consecutive memory locations. For this reason, the IPA register is loaded with a pretranslated copy of the address of the first longword of an instruction stream and then incremented to obtain consecutive longword addresses of instructions. This eliminates the unnecessary translation process for each consecutive longword of instructions. Once loaded the IPA register is incremented coincidentally with the VIBA register.

The IPA register is loaded under microcode control by the READ. V. NEWPC command. This is normally done when a macrocode transfer of control occurs resulting from a branch, jump, or jump to subroutine.

When the IPA register is incremented across a page boundary, an auto-reload feature automaticaly reloads the IPA register with the next address of the instruction without the need of microcode control. For this, the IPA is reloaded with a translated copy of the address in the VIBA register from the TB matrices (or an untranslated address if mapping is not enabled). The reload is controlled by hardware sequencing logic which is capable of starting the reload during any ALLOW.IB.READ microcycle. The actual reload is executed in the following microcycle. If the following microcycle requests a memory operation, the memory operation is stalled for 200 ns.

If a TB miss, access violation, or parity error occurs during the IPA reload, a cancel signal is asserted. This signal indicates that the IPA address is invalid. In this case the microcode is notified of the error(s) when the instruction buffer runs out of data. The appropriate microtrap routine is then executed.

If the CPU tries to use the VA register during an IPA reload, the CPU is stalled until the current ALLOW.IB.READ is complete.

As seen in Figure 2-7, the appropriate page address is enabled from the VA register, IPA register, or TB matrices to the PA bus via the PA mux. Similarly, the byte offset (which is not used in the translation process) is selected from the VA register for a data reference or the IPA register for an instruction reference.

Table 2-1 Page Accessibility for Each Processor Mode

Protect Code		Access Allowed			
Hex	**Binary**	**K**	**E**	**S**	**U**
0	0000	None	None	None	None
1	0001	(Reserved)	(Reserved)	(Reserved)	(Reserved)
2	0010	RW	None	None	None
3	0011	R	None	None	None
4	0100	RW	RW	RW	RW
5	0101	RW	RW	None	None
6	0110	RW	R	None	None
7	0111	R	R	None	None
8	1000	RW	RW	RW	None
9	1001	RW	RW	R	None
A	1010	RW	R	R	None
B	1011	R	R	R	None
C	1100	RW	RW	RW	R
D	1101	RW	RW	R	R
E	1110	RW	R	R	R
F	1111	R	R	R	R

K = Kernel
E = Executive
S = Supervisor
U = User

R = Read Only
RW = Read or Write

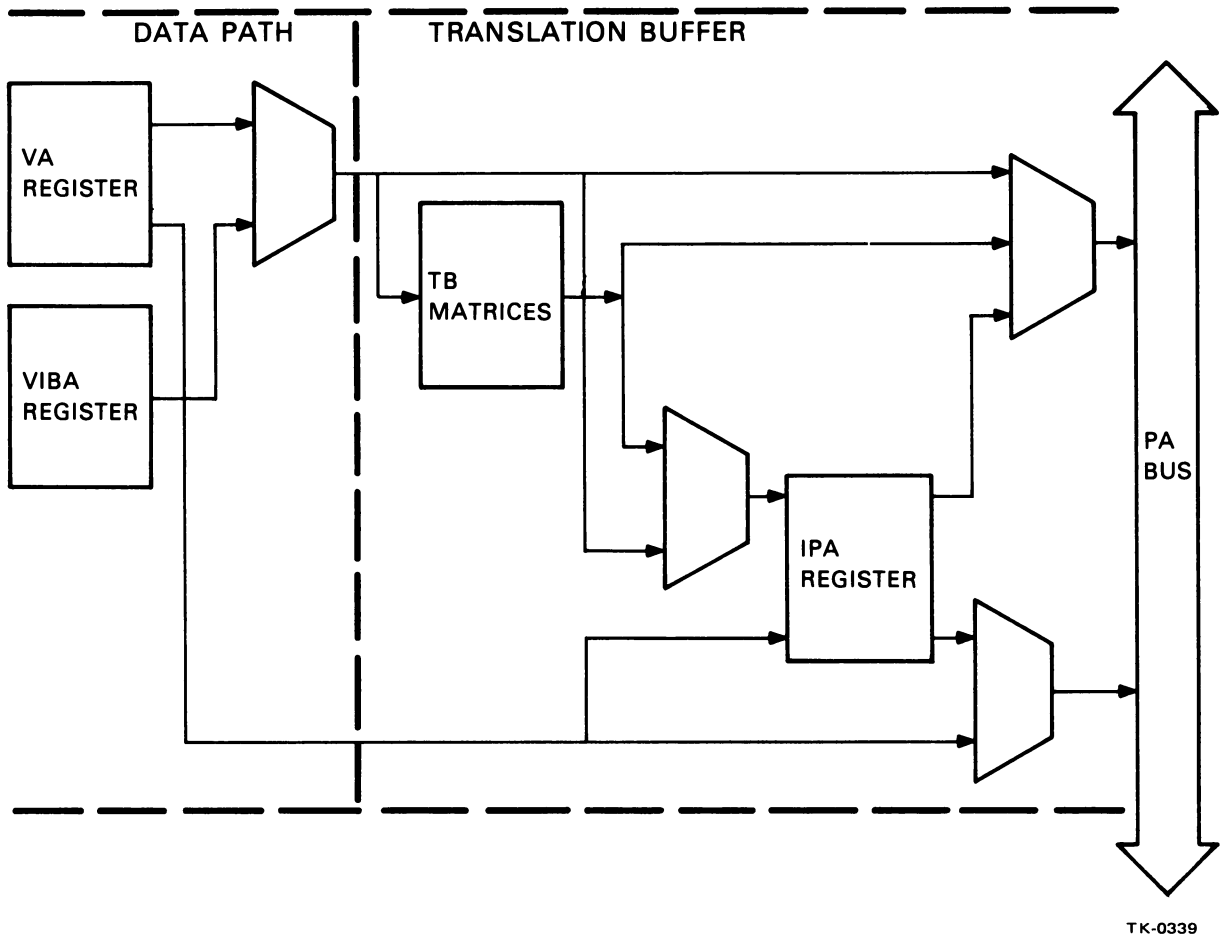


Figure 2-7 Translation Buffer - Simplified Block Diagram

2.1.5 TB Logic Description

Figure 2-8 and 2-9 contain a block diagram of the logic associated with the TB address matrix. The following paragraphs describe this logic.

As seen in Figure 2-8, the TB address matrix and associated logic is located on the Cache address matrix board (CAM). Here, the index portion of an incoming address [DDP VAMX (31,13:09)] selects a location in both groups of the TB address matrix. The contents of these locations are enabled to parity checking logic and comparators. The tag portion of the incoming address [DEP VAMX (30:14)] is buffered and also input to the comparators as TBMA VA MUX (30:14) BUFF. In addition, two parity bits are generated for the incoming tag and input to the comparators. If the incoming tag and parity bits match the contents of the indexed location of either group, a TB hit occurs. This indicates that the corresponding location in the data matrix contains the page frame number for the address translation. The protection codes and modify bits of the matrix locations are input to a mux in anticipation of a TB hit. The output of the mux is connected to protection decode logic on the TB data matrix board (TBM). If a TB hit occurs and parity is good, CAMU TB GRP0 MATCH or CAMU TB GRP 1 MATCH is generated and sent to the data matrix along with the corresponding protection code.

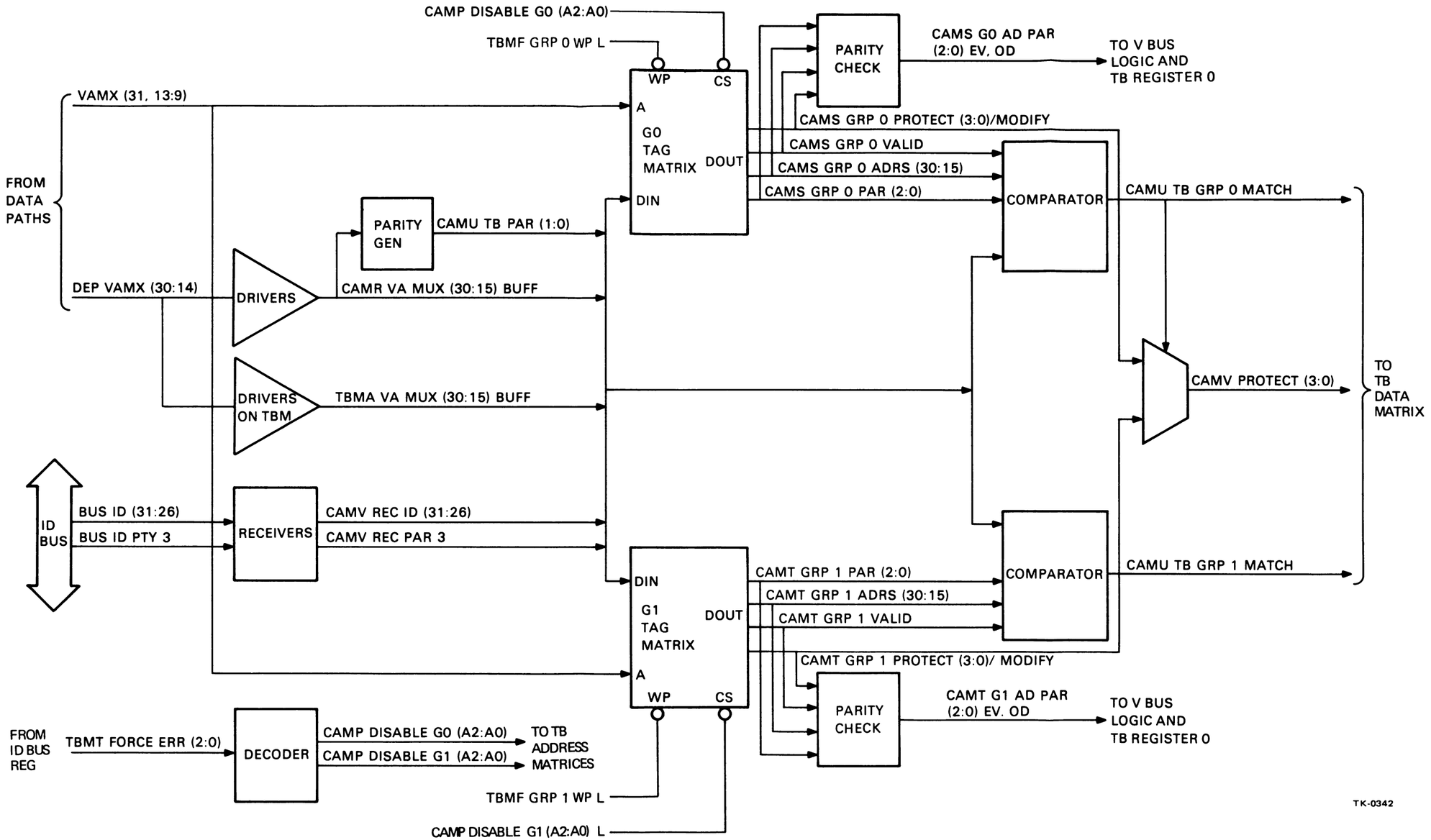


Figure 2-8 Translation Buffer Address Matrix (on Cache Address Matrix Board)

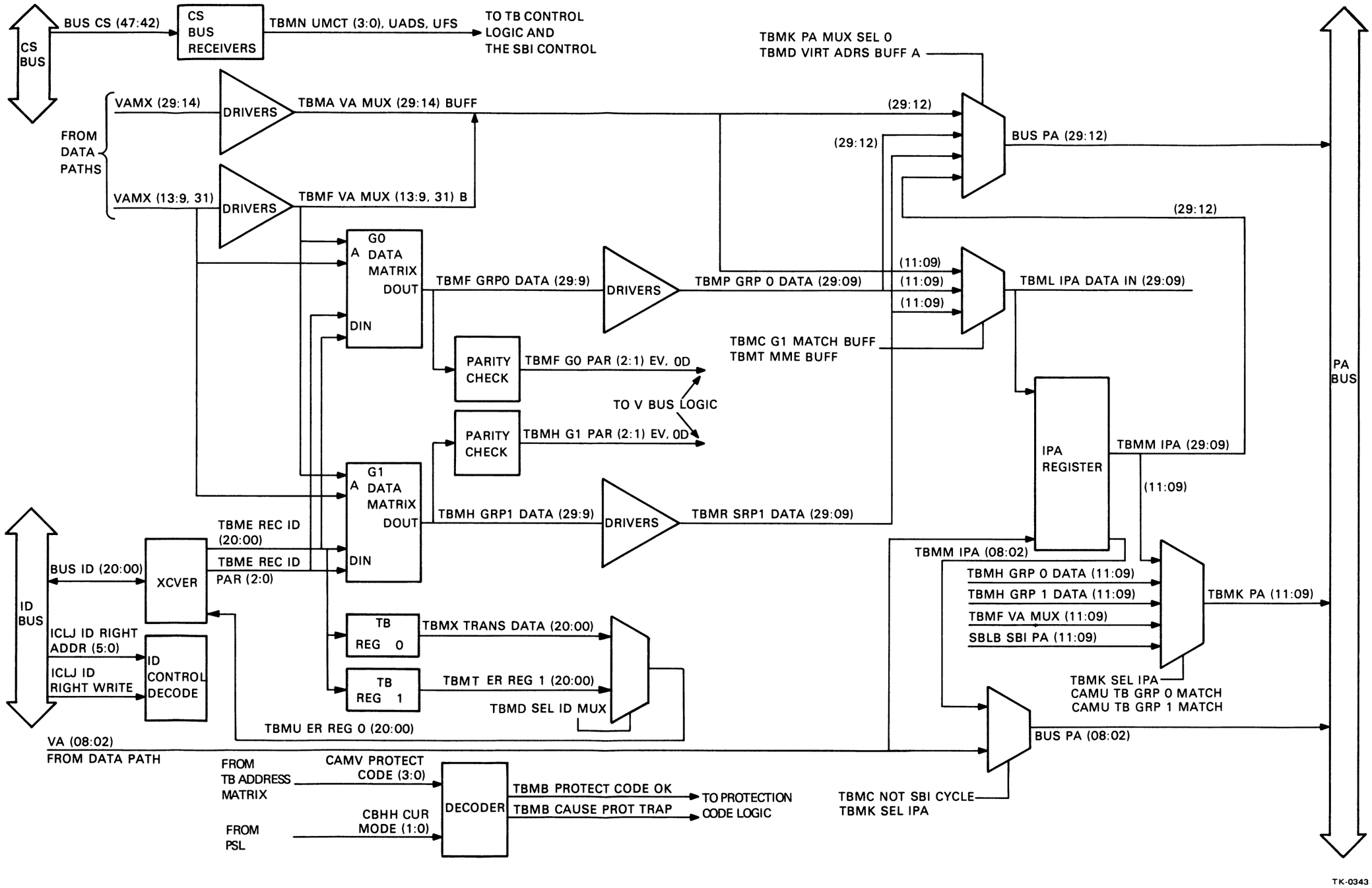


Figure 2-9 Translation Buffer Data Matrix

If a TB miss occurs (no tag match), the PTE is fetched from main memory and placed on the ID bus under microcode control. The V bit, protect code, and M bit are then received from the ID bus and input to the address matrix. (Likewise, the page frame number is received from the ID bus and stored in the corresponding data matrix location.) With the index still asserted by the data path, TBMF GRP 0 WPL or TBMF GRP 1 WP L is asserted with ID address 10 by the microcode to write the information into a randomly selected group of the address matrix. A parity bit for this information is also received from the ID bus and stored in the matrix location. Note a parity check on this information is not executed until the information is read from the matrix.

When the index of an incoming address is sent to the address matrix, a copy is also sent to the data matrix in anticipation of a TB hit. Figure 2-9 illustrates the logic associated with the TB data matrix. If a TB hit occurs and parity is good, the protection code [CAMV PROTECT CODE (3:0)] from the address matrix is enabled to decode logic on the TBM board. If access to the page is not allowed, TBMB CAUSE PROT TRAP is generated and a microtrap occurs.

In addition to the protection check, the contents of the indexed location in the corresponding group of the data matrix are buffered and transferred to the PA mux logic. The PA mux logic and IPA logic select the data matrix, IPA register, or data path itself as the source of the physical page address. The operation of the PA mux logic and IPA logic is described in Paragraph 2.1.4. Note that the output of the data matrix is also sent to parity checking logic.

As mentioned previously, if a TB miss occurs, the PTE is fetched from main memory and transferred over the ID bus to be stored in the TB. Portions of the PTE are received from the ID bus and stored in the address matrix. Similarly, the page frame number (physical page address) is received from the ID bus and stored in the data matrix. When the PFN is received, three parity bits [TBME REC ID PAR (2:0)] are generated and also stored.

2.1.5.1 TB Data Register – The TB Data Register is addressable over the ID bus. When the microcode has retrieved a PTE from memory during a TB miss, the PTE is placed on the ID bus with the proper address and control information. The ID bus control logic on the TBM board decodes the ID bus address and enables the data matrix appropriately to latch the PTE.

2.1.5.2 TB Register 0 and 1 – The ID transceivers used to receive the PFN from the ID bus are also used to transmit information from TB Register 1 and TB Register 0. These registers are readable over the ID bus and contain TB related information and status. Figure 2-10 illustrates each register format. Table 2-2 describes the bits of TB Register 0 and Table 2-3 describes the bits of TB Register 1.

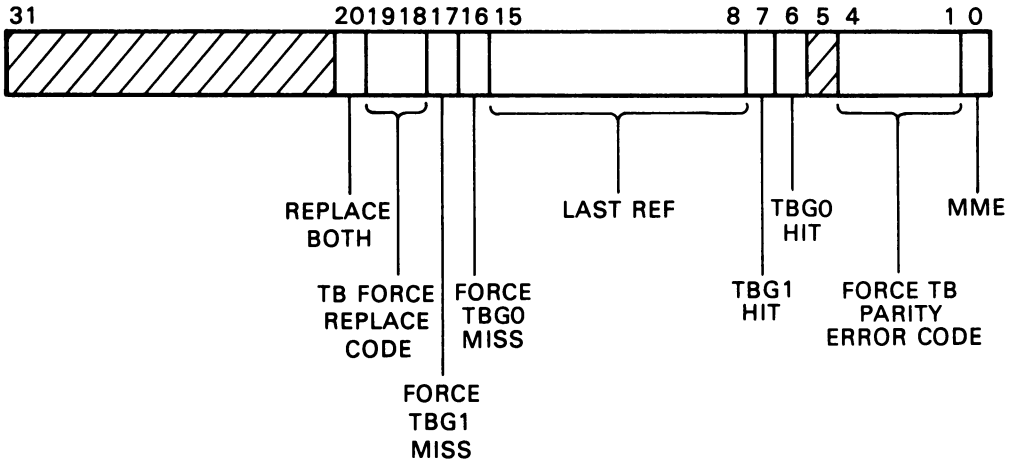
2.2 CACHE DESCRIPTION

2.2.1 General Cache Concepts

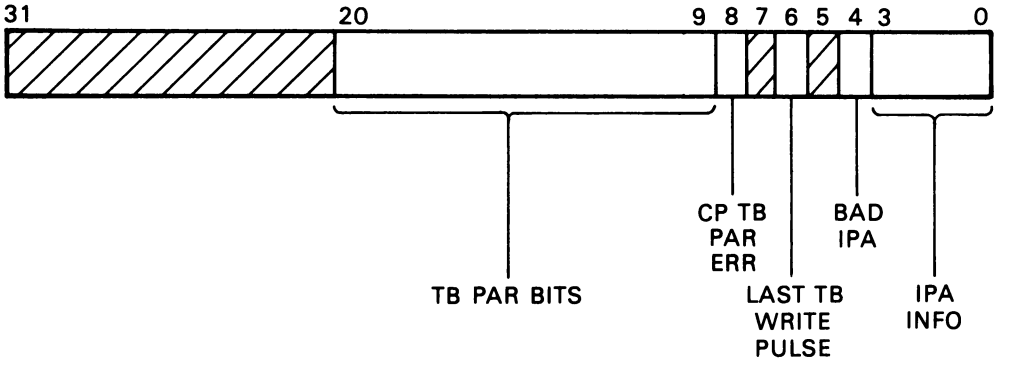
This section explains the purpose of cache memory systems and describes various methods used to implement such systems. Parameters and strategies involved in cache memory design are introduced, described, and analyzed to facilitate the reader's understanding of the specific Cache implemented in the KA780.

2.2.1.1 Overall Organization of a Cache Memory System – The cache memory system is intended to simulate a system having a large amount of fast memory. To do this, the cache system relies on a small amount of very fast memory (the cache), a large amount of slower memory (the main memory), and the statistics of program behavior.

TB REGISTER 0



TB REGISTER 1



TK-0350

Figure 2-10 TB Registers 1 and 0

Table 2-2 TB Register 0 Bit Assignment

Bit	Function	Description
20	Replace Both	If set, both groups in the TB address and data matrices are written with data from the TB Data register. This bit is normally used when clearing the TB and is read/write.
19,18	TB Force Replace Code	Selects a group in the TB address and data matrices for a TB write. The contents of the TB Data register are written in the selected group. 19 | 18 | Selected Group 0 | 0 | Random Group 0 | 1 | Group 0 1 | 0 | Group 1 1 | 1 | Unused These bits are read/write.
17	Force TB G1 Miss	If set, prevents any TB hits in group 1. This bit is read/write.
16	Force TB G0 Miss	If set, prevents any TB hits in group 0. This bit is read/write.
15:8	Last Reference	These bits contain the following information about the most recent memory request by the microcode. These bits are read-only. 15 UFS 14 UADS 13 UMCT3 12 UMCT2 11 UMCT1 10 UMCT0 09 IB WCHK from the instruction buffer 08 The cycle was delayed one cycle by an auto-reload of the IPA.
7	TB G1 Hit	If set, indicates a TB hit in group 1. This bit is the latched output of the group 1 address comparator. It is used for diagnostics and is read-only.
6	TB G0 Hit	If set, indicates a TB hit in group 0. This bit is the latched output of the group 0 address comparator. It is used for diagnostics and is read-only.

Table 2-2 TB Register 0 Bit Assignment (Cont)

Bit	Function	Description
4:1	Force TB Parity Error Code	Forces a parity error in the group selected as shown here. These bits are read/write.
0	Memory Management Enable	If not set, all addresses are placed on the PA bus directly from the data path. The TB miss, parity, page protection, M-bit, and page boundary microtraps are also disabled. This bit is read/write.

Bits 4 3 2 1	Group	Adrs/Data Matrix Byte
0 0 0 0	–	–
0 0 0 1	–	–
0 0 1 0	0	D0
0 0 1 1	0	D1
0 1 0 0	0	D2
0 1 0 1	1	D0
0 1 1 0	1	D1
0 1 1 1	1	D2
1 0 0 0	0	A0
1 0 0 1	0	A1
1 0 1 0	0	A2
1 0 1 1	1	A0
1 1 0 0	1	A1
1 1 0 1	1	A2
1 1 1 0	–	–
1 1 1 1	–	–

Table 2-3 TB Register 1 Bit Assignment

<table>
<tr><th>Bit</th><th>Function</th><th>Description</th></tr>
<tr><td>20:9</td><td>TB Par Bits</td><td>These bits are loaded when a TB parity error occurs while the IPA is being loaded or TB parity traps are enabled. If set, a parity error exists in the corresponding byte as follows:
<table>
<tr><th>Bit</th><th>Group</th><th>Adrs/Data Matrix Byte</th></tr>
<tr><td>20</td><td>1</td><td>D2</td></tr>
<tr><td>19</td><td>1</td><td>D1</td></tr>
<tr><td>18</td><td>1</td><td>D0</td></tr>
<tr><td>17</td><td>0</td><td>D2</td></tr>
<tr><td>16</td><td>0</td><td>D1</td></tr>
<tr><td>15</td><td>0</td><td>D0</td></tr>
<tr><td>14</td><td>1</td><td>A2</td></tr>
<tr><td>13</td><td>1</td><td>A1</td></tr>
<tr><td>12</td><td>1</td><td>A0</td></tr>
<tr><td>11</td><td>0</td><td>A2</td></tr>
<tr><td>10</td><td>0</td><td>A1</td></tr>
<tr><td>09</td><td>0</td><td>A0</td></tr>
</table></td></tr>
<tr><td>8</td><td>CP TB Par Err</td><td>If set, this bit indicates that the TB has requested a TB parity error microtrap. This bit is read-only and is cleared by any write to TB Register 1.</td></tr>
<tr><td>6</td><td>Last TB Write Pulse</td><td>This bit is read-only and indicates which TB group was most recently written. If both groups were modified, it is indeterminant.

1 = Group 1
0 = Group 0</td></tr>
<tr><td>4</td><td>Bad IPA</td><td>If set, the information in the IPA register and the IPA info bits of TB Register 1 are invalid. This bit is read-only and is set by counting across a page boundary (or by FLUSH), and cleared by loading the IPA register.</td></tr>
<tr><td>3:0</td><td>IPA Info</td><td>These bits contain the following information about the most recent loading of the IPA register.

3 MISS, A TB miss occurred during the loading.

2 PARITY, A parity error occurred during the loading.

1 PROTECT, A protection violation occurred during the loading.

0 AUTO LOAD, The loading was automatic and not the result of READ.V.NEWPC.

If set, the condition exists. These bits are read-only.</td></tr>
</table>

The basic idea is to store some data in the fast memory and some in the slow memory. If it can somehow be arranged that data is in the fast memory when the processor needs it, the program will execute quickly, slowing down only occasionally for main memory operations. Conventional, mixed MOS-core systems attempt to achieve this goal by having the programmer guess beforehand which sections of his program should go in each memory. This is often awkward and usually only moderately successful. The cache memory system tries to achieve the same goal by automatically, dynamically shuffling data between the two memory types in a way which gives a high probability that useful data will be in the fast memory. All of the following discussions of cache organizations and strategies are intended to show implementable methods of shuffling data, so that the data most likely to be needed next will be in the fast memory instead of the slower main memory.

2.2.1.2 Program Locality – A cache memory works because it can usually predict successfully which words a program will require soon. If programs used words completely at random from all of memory, it would be impossible to predict which words would most likely be needed next. Under these circumstances, a cache memory system could perform no better than a conventional mixed memory system with a small amount of bipolar memory.

Fortunately, programs do not generate random addresses. Instead, programs have a tendency to make most accesses in the neighborhood of locations accessed in the recent past. This is the basis of the principle of program locality. The fact that programs display this type of behavior makes cache memory systems possible.

An understanding of why the principle of program locality is true can be obtained by examining the small scale behavior of typical program data structures. Code execution itself generally proceeds in straight lines or small loops; the next few accesses are most likely to be within a few words, ahead or behind. Stacks grow and shrink from one end, with the next few accesses near the current top. Character strings and vectors are often scanned through sequentially.

The principle of program locality is a statement of how most programs tend to behave, not a law which all programs always obey. Jumps in code sequences, seemingly random access of symbol tables by assemblers, and context switching between programs are examples of behavior which can adversely affect the locality of addresses generated by a processor. The process of guessing which words a program will reference next can never be completely successful. The percentage of correct guesses is a statistical measure affected by the size and organization of the cache, the algorithms it uses, and the behavior of the program driving it.

2.2.1.3 Block Fetch – The principle of program locality states that for the cache to have the best chance of having the word the program needs next, the cache should have words near those recently used. The basic method of accomplishing this is the block fetch. When the cache controller finds it necessary to move a word of data from slow memory to fast memory because the data was not in the fast memory when needed, the controller will move not just the word required, but a block of several adjacent words at once. Typically, the block will contain one (degenerate case), two, four, or eight words starting on an even block boundary.

The block fetch can provide either look-behind, look-ahead, or both, depending on the position of the originally requested word within the block. Since many important generated address sequences (e.g., most code) tend to move in increasing order, the originally requested word is usually the first in the block, so the block fetch generally provides look-ahead.

The block size is one of the most important parameters in the design of a cache memory system. If the block size is too small, the system will have insufficient look-ahead and performance will suffer slightly, particularly for programs which do not contain many loops. Also, as will be discussed later, small block sizes require the system to store more addresses than large blocks, for the same total memory size.

If the block is too large, there may not be room for enough blocks in the cache to provide for adequate look-behind. Large blocks also tend to mean more memories operating in parallel within the slow memory and therefore wider buses between slow and fast memory, resulting in increased cost. As the block gets larger, each additional word in the block is less likely to be useful, since it is further from the originally requested word and less likely to be needed soon by the program. It has been found empirically that while a block size of two words increases memory system performance dramatically, further increases in block size produce much smaller improvements which are seldom worth implementing.

2.2.1.4 The Fully Associative Cache – If a cache memory system was designed so that the fast memory held one contiguous block of 1000 words, it would fail miserably. Most programs make reference to code segments, subroutines, stacks, lists, and buffers located in scattered parts of the whole address space. Ideally, a 1000-word cache would hold the 1000 words the controller estimated as most likely to be needed, no matter how scattered these words were throughout the address space of main memory.

Since there would be no relation of all the addresses of these thousand words to each other or to any single register or mapping function, each of the 1000 data words in the fast memory would have to carry its address with it. Then, when the processor requested a word from memory, the cache would simply compare (associate) the address from the processor with each of the thousand addresses of words in the fast memory. If a match were found, the data for that address would be sent to the processor. This is the principle of an associative memory (Figure 2-11).

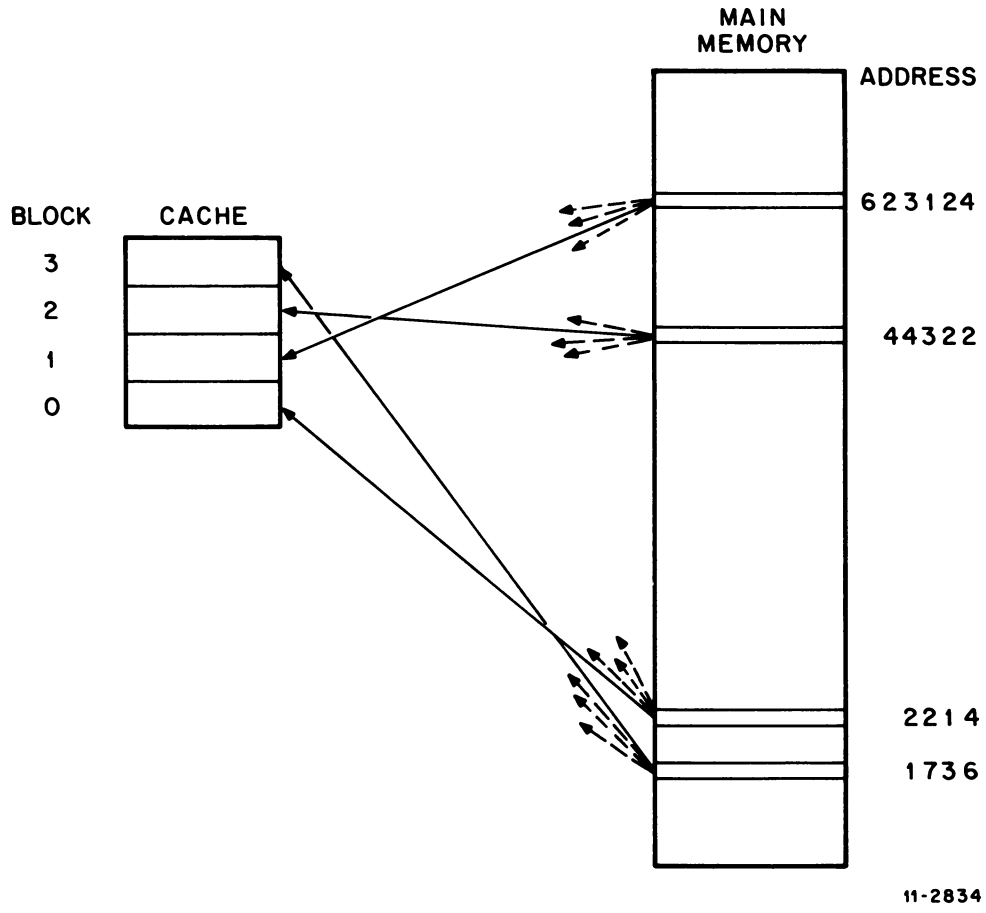


Figure 2-11 A Fully Associative Cache Memory System

This system, called fully associative because the incoming address must be compared (associated) with all the stored addresses, gives the cache controller maximum flexibility in deciding which words it wants in fast memory; i.e., any words at all until the memory is full. Unfortunately, 1000 address comparisons would be unacceptably slow and/or expensive. One of the basic issues of cache organization is how to provide minimum restrictions on what groups of words may be present in fast memory, while limiting the number of address comparisons required.

2.2.1.5 The Direct Mapping Cache – At the opposite extreme from the fully associative cache is the direct mapping cache. Instead of one address comparison on every block, the direct mapping cache requires only one address comparison.

The many address comparisons of the fully associative cache are necessary because any block from main memory can be placed in any block of fast memory. Thus, every block of fast memory must be checked to see if it has each requested address. The direct mapping cache allows each block from main memory only one possible location in fast memory (Figure 2-12). Consider each incoming address as having three parts. The first part (address field) starts at bit 0 and contains enough bits to specify which byte out of a block is being requested. The next field, called the index field, starts where the first field leaves off and contains enough bits to specify any block in fast memory. The third field, called the address field, contains the rest of the bits.

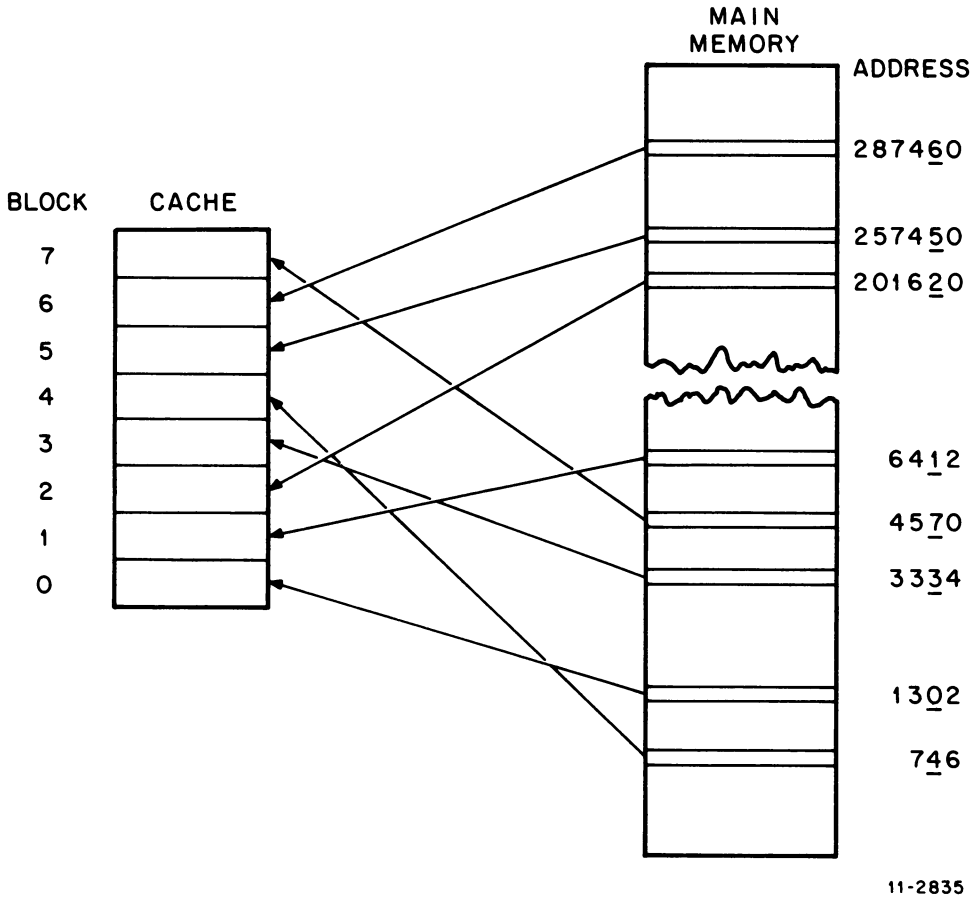


Figure 2-12 A Direct Mapping Cache Memory System

As an example, consider an 18-bit PDP-11 byte address as input to a 256-word, 4-word-per-block, direct mapping cache. (This cache would thus be 4 words wide and 64 blocks deep. Assuming 4 words per block, allows us to break down the address conveniently, using octal notation.) As illustrated in Figure 2-13, the word field in this case comprises bits 2, 1, and 0, where bit 0 indicates the byte and bits 2 and 1 indicate the word. The index field comprises bits 8 through 3, and indicates the block. The address field comprises bits 17 through 9.

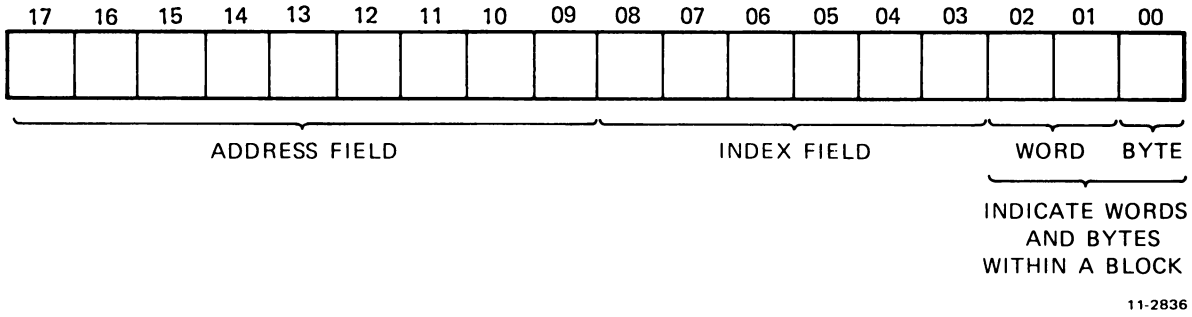


Figure 2-13 18-Bit Byte Address Breakdown
(4 Words per Block, 64 Blocks)

If the processor requests word 274356, the cache controller looks at the address which goes with the information currently in block number 35 in fast memory. If this address field is 274, the controller sends the third word in that block to the processor. If the stored address field is not 274, the controller must fetch block 27435 (located at address 274350) from main memory, transmit the third word in the block to the processor, load the block into block 35 of fast memory, replacing whatever was there previously, and change the address field stored with block 35 to 274.

Any address whose index field is 35 will be loaded in block 35 of fast memory, and therefore this is the only place the cache controller has to look if the processor requests the data from an address whose index field is 35.

Notice also that only the address field of the address need be stored with each block, because only the address field of the address is required for comparison. The index field need not be compared because anything stored in fast memory block 35 has an index field of 35. The word field need not be compared because if the block is there, every word in the block is there.

This is how the direct mapping cache uses inexpensive direct addressing of fast memory to eliminate almost all comparison operations.

Of course there are disadvantages to this simple scheme. If the processor in the example makes frequent references to both locations 274356 and location 6352, there will be frequent references to slow memory, because only one of these locations can be in the cache at one time. Fortunately, this sort of program behavior is infrequent, so that the direct mapping cache (although offering significantly poorer performance than fully associative) is adequate for some applications. Usually the system of choice is a compromise between a direct mapping cache and fully associative cache, called the set associative cache.

2.2.1.6 The Set Associative Cache – The set associative organization is a compromise between the extremes of fully associative and direct mapping. This type of cache has several directly mapped groups (Figure 2-14). For each index position in fast memory there is not one block, but a set of several – one in each group. (The set of blocks corresponding to an index position is called a "set.") A block of data arriving from main memory can go into any group at its proper index position.

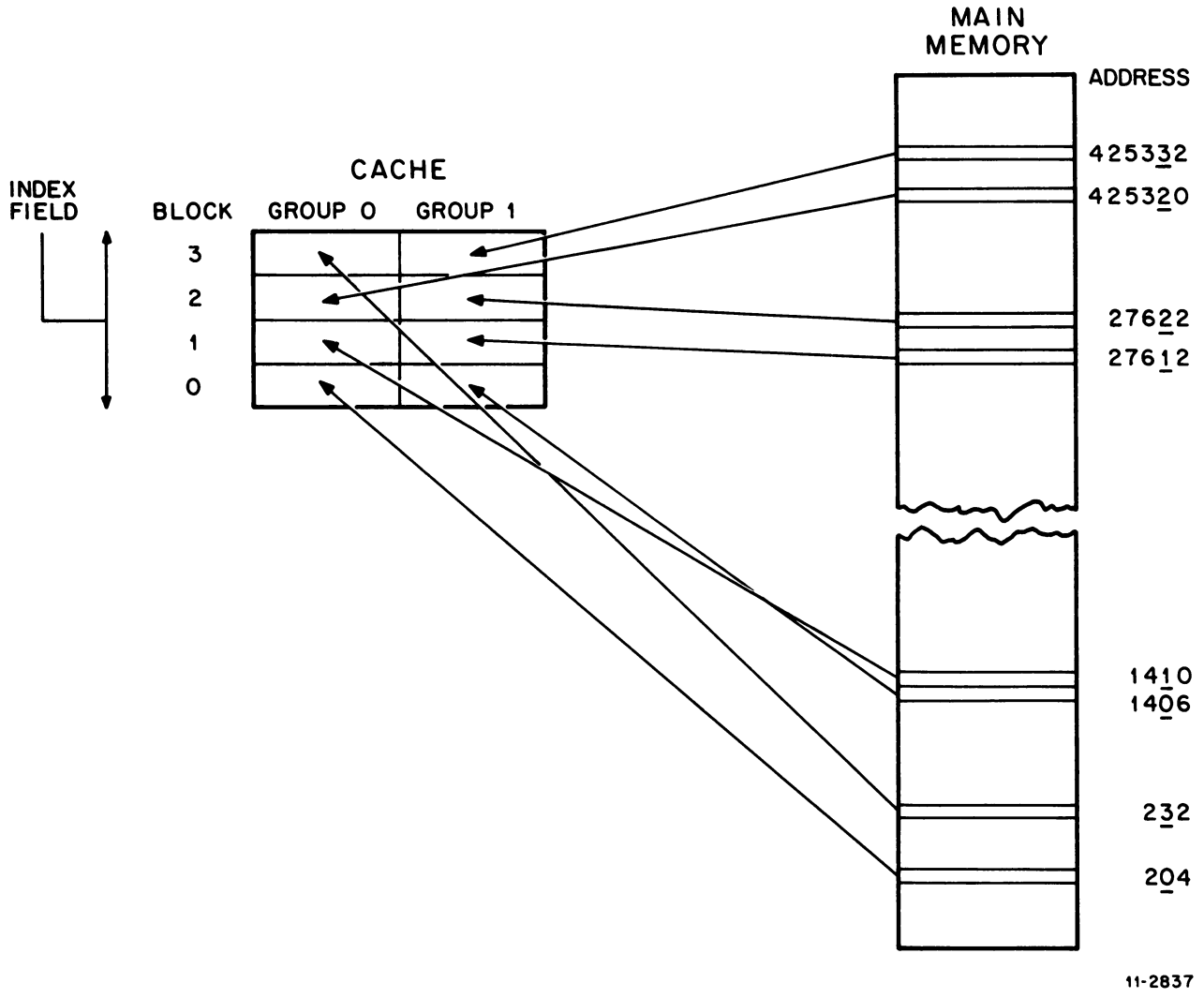


Figure 2-14 Set Associative Cache Memory System (Two-Way)

Since there are several places for data with the same index field in their addresses to be stored, the type of excessive main memory traffic possible in a direct mapping organization is less likely to occur. This gives a set associative cache higher performance. In fact, a four-way set associative cache (four groups) will normally perform very nearly as well as a fully associative cache.

The price that is paid for higher performance is some increase in complexity. There are several places in fast memory where any given piece of data can be stored, so the controller must do several compares (i.e., must associate) to determine in which place (if any) the requested data is located. The number of times it must compare (associate) is of course equal to the number of groups, usually two, three, or four. A set associative cache can be classified as an n-way set associative cache, where "n" is the number of compares performed (i.e., the number of groups).

Another aspect of the increased complexity becomes apparent when a block of fast memory must be overwritten. There are now several locations in fast memory where the new data from main memory may be written (one in each group), so the controller must have some means of deciding which block will be overwritten. The decision could be made using any of the following considerations.

Least Recently Used (LRU) – The block least recently used is replace.

First-In, First-Out (FIFO) – The block which has been stored the longest time is replaced.

Random – Blocks are replaced in a random manner.

A replacement strategy based on LRU or FIFO information requires the storage of LRU or FIFO bits, along with the address fields in the address memory and the logic necessary to generate and decode these bits. The random strategy is far easier and cheaper to implement, yet provides performance only slightly lower than that obtainable by the other strategies.

The extra performance of a set associative cache usually justifies the slightly extra complexity of at least two-way associativity in all but low performance applications.

2.2.1.7 Write-Through and Write-Back – Assume that the following sequence of events occurs. First, the processor does a read of location 200, resulting in the block with this address being copied into fast memory. Then the processor writes new data into location 200, updating this location in fast memory. Next the processor does a reference which causes the cache controller to overwrite the block in fast memory containing location 200. If the processor reads location 200 again, the obsolete data in main memory will be loaded into fast memory. This is unacceptable, and two methods have been devised to deal with this problem of stale data. The methods are called write-through and write-back.

With write-through, whenever a write reference occurs, the data is not only stored in fast memory, but is also immediately copied into main memory. This means that the main memory always contains a valid copy of all data.

The advantages of write-through are its relative simplicity and the fact that the main memory always has correct data. The primary disadvantage is some reduction of speed due to the need to access the slow memory on every write reference. This is offset somewhat by the fact that write references are a small fraction of all references to memory. In addition the cache does not have to wait for the main memory to finish before starting the next cycle.

The other method of handling the stale data problem in a cache system is called write-back. Under this method, data written by the processor is only stored in the fast memory, leaving the main memory unaltered and obsolete. A bit in the address field of the block in fast memory, called the altered bit, is set to indicate that this block contains new information. When the controller wants to overwrite a block of fast memory, the altered bit is inspected first. If this bit is set, the controller must write the block into main memory before overwriting it.

The primary advantage of write-back is higher performance. For almost any program, the number of times an altered block must be copied into main memory is less than the number of write references, so write-back is noticeably higher performance than write-through. One disadvantage of write-back is increased complexity. A write-back system must have the ability to regenerate addresses from tags and the extra sequencing logic to do double cycles.

Another disadvantage of write-back is the power fail problem. When power fails, fast memory will be holding the only valid copies of some arbitrary set of locations. If these are not copied into main memory, they will be lost. Since there is no way of knowing which locations were lost, the entire memory must be considered volatile. If main memory is volatile anyway, there is no problem; otherwise, steps must be taken. One possibility is to require the power fail program to do a sequence of reads calculated to ensure that every block in the cache has been overwritten. A more reliable, but more expensive system would automatically ensure that all altered blocks are copied into main memory, after the program halts, but before power disappears.

2.2.2 The Cache of the KA780

The following paragraphs describe the Cache which has been implemented in the KA780. The reader should be familiar with the cache concepts described in the previous paragraphs.

As mentioned in the Introduction, the Cache used in the KA780 is two-way set associative. It consists of two groups of 1024 longword entries. Thus the total capacity of the fast memory is 2K longwords. The Cache is implemented using a random replacement strategy. The write strategy is write-through and not-write-allocate.

One address is stored for each pair of longwords (one address for each quadword). If a cache hit occurs, the appropriate longword is transmitted to or from Cache over the MD bus. If a cache read miss occurs, the quadword containing the requested data is brought from memory via the SBI Control and placed in Cache. Coincidentally the appropriate longword is transmitted to the requestor. If a cache write miss occurs, the appropriate longword is updated in main memory only. For all writes (hit or miss), a memory operation is executed.

2.2.3 Cache Matrix Structures

The organization of the Cache data matrix (CDM) of the KA780 is illustrated at the bottom of Figure 2-15. Note that the CDM is divided into two equal groups (Group 0 and Group 1). Each group contains 512 blocks or 1024 longwords. Four parity bits are also stored with each longword of data. Bits (11:3) of the incoming address index into the CDM and select a block from group 0 and the same block from group 1. Bit 2 of the incoming address enables either the low (even) longword and its parity bits or the high (odd) longword and its parity bits from both groups to a multiplexer. One of these longwords is selected if a hit is detected in the Cache address matrix. The longword selected is from the group in which the hit occurred.

The organization of the Cache address matrix is also illustrated in Figure 2-15. Its structure is determined by that of the Cache data matrix. Note that it also is divided into two equal parts: Group 0 corresponding to Group 0 in the data matrix and Group 1 corresponding to Group 1 in the data matrix. Since a tag must be stored to identify each block in the data matrix, 1024 locations are required; 512 in group 0 and 512 in group 1. Each location contains a valid bit, 17 tag bits, and 3 parity bits. The valid bit indicates the integrity of the data quadword in the CDM corresponding to the tag. Just as in the data matrix, bits (11:3) select a tag from both groups. The two tags are read from the address matrix and compared with the tag field (bits 29:12) of the incoming address. If either comparison results in a match, if the corresponding valid bit is set, and if no parity error is detected, a hit occurs. A read hit selects the corresponding longword and parity bits in the CDM for output from the matrix. A write hit enables a write pulse to the corresponding group in the CDM.

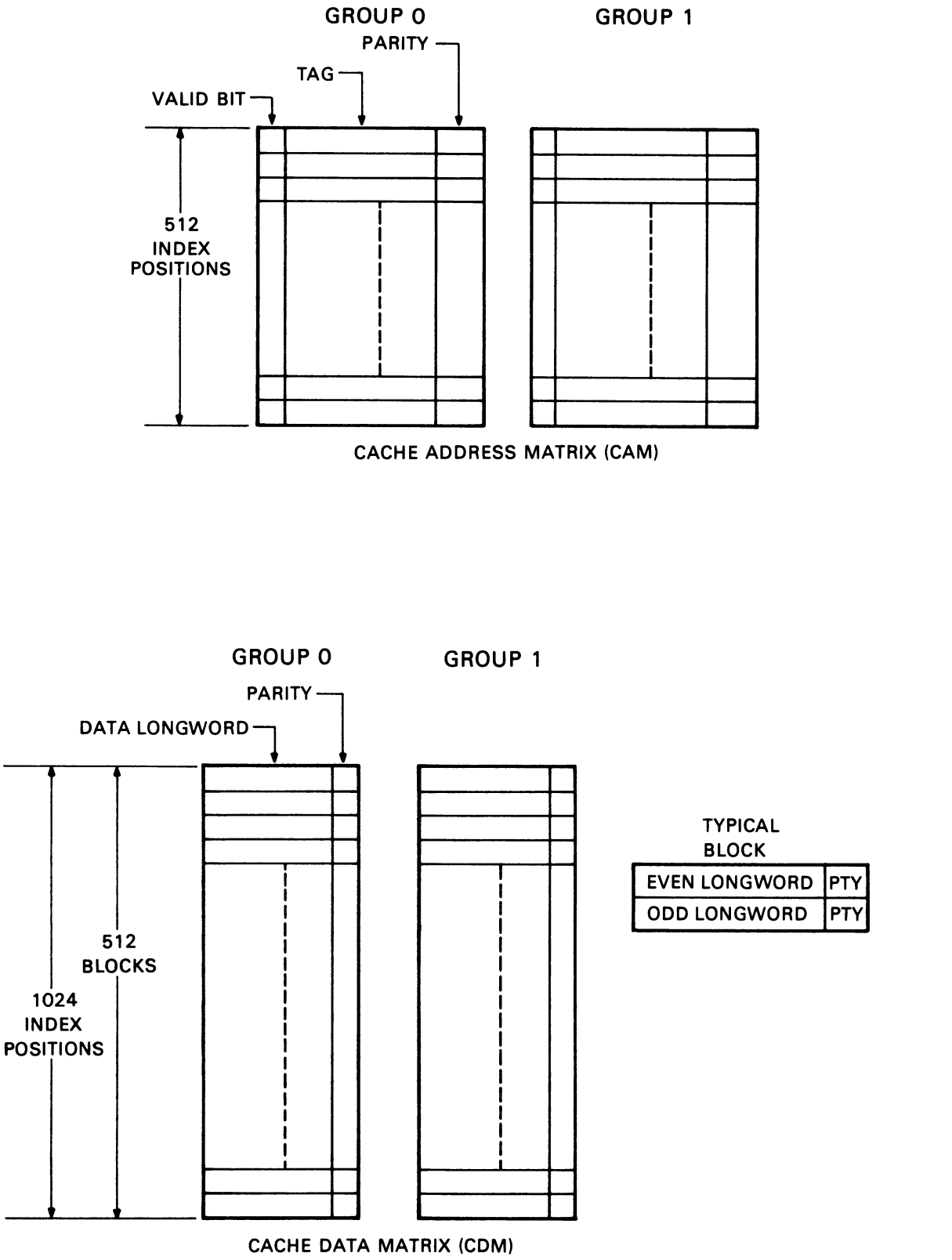


Figure 2-15 Cache Matrix Structures

2.2.4 Cache Logic Description

Figures 2-16 and 2-17 contain a block diagram of the logic associated with the Cache address and data matrices. The following paragraphs describe this logic and Cache operation in general.

2.2.4.1 Address Path - As seen in Figure 2-15, for any Cache operation the address is latched from the PA bus and transferred to the cache address matrix for a data look-up. The index portion [BUS PA (11:03)] is buffered and sent to the address matrix where it selects a location in both groups. The corresponding tags, valid bits, and parity bits at these locations are enabled to comparators along with the latched tag, valid bit, and parity bits from the PA bus. A match between the tag from the PA bus and the tag from either group indicates the corresponding location in the data matrix contains the data for the operation. If a tag and parity match occurs and the valid bit is set, CAMK G0 MATCH or CAMK G1 MATCH is generated indicating a cache hit. These signals control the MD mux in the Cache data matrix. The absence of both signals indicates a Cache miss; that is, the requested data is not in the data matrix.

When a read miss occurs, the indexed block of Group 0 or Group 1 is rewritten with data from main memory. The group is arbitrarily selected by logic in the SBI Control. This logic arbitrarily generates SBLN MISS DATA G0 or SBLN MISS DATA G1 H to enable a write pulse (CAMP G0 WRITE EN H or CAMP G1 WRITE EN H) to the corresponding group.

2.2.4.2 Data Path - When the address is latched from the PA bus in the Cache address matrix, it is also latched to select a location in the Cache data matrix (Figure 2-17). This is done in anticipation of a Cache hit. Bus PA 02 is also latched for the Cache data matrix to select a longword of the quadword block. Data may be read from or written into the matrix via the MD bus.

During a read the data longword at the indexed location in both groups of the data matrix are enabled to the MD mux. Both longwords are parity checked. A match signal from the Cache address matrix selects the output from the corresponding G0 or G1 for transfer to the MD bus.

During a read miss, the quadword containing the requested data is fetched from main memory by the SBI Control (two successive SBI cycles). The SBI Control performs SBI protocol checks and a parity check on the data. If a parity error is detected, the retrieved data is not placed on the MD bus and an SBI timeout occurs. Only data retrieved with good parity is placed on the MD bus for transfer to Cache and the requestor (data paths or instructions buffer). When the longword containing the requested data is placed on the MD bus, the requestor is notified. If the data was requested by the data path, the CPU is stalled (Paragraph 2.2.4.5) until the requested data is placed on the MD bus.

To update Cache, the indexed block of Group 0 or Group 1 is also rewritten with data from the MD bus. The group is arbitrarily selected by the SBI Control which generates SBLN MISS DATA G0 H or SBLN MISS DATA G1 H. These signals enable write pulses to the corresponding group.

For a write, the data longwords are latched from the MD bus by Cache along with their parity bits. BUS MD BYTE (3:0) MASK H is also latched to provide the proper selection of write pulses. The data and parity bits are written into the indexed location of the group selected by CAMP G0 WRITE EN H or CAMP G1 WRITE EN H. One of these signals is produced by a tag match in the address matrix (write hit). Whether or not a tag match occurs, the data and write pulses are also simultaneously latched by the SBI Control which updates the location in main memory in accordance with the write-through strategy. The CPU is not stalled during this SBI cycle unless the SBI write buffer is full with a previous request.

The first longword of the Cache update is always written into Cache as invalid data; that is, the valid bit is not set. When the second longword is transferred, however, the valid bit is set for both longwords, provided a parity error did not occur. This prevents the writing of both valid and invalid data in the same block (only one valid bit describes both longwords of the block). If a parity error or access violation occurs for either longword, the valid bit remains unset.

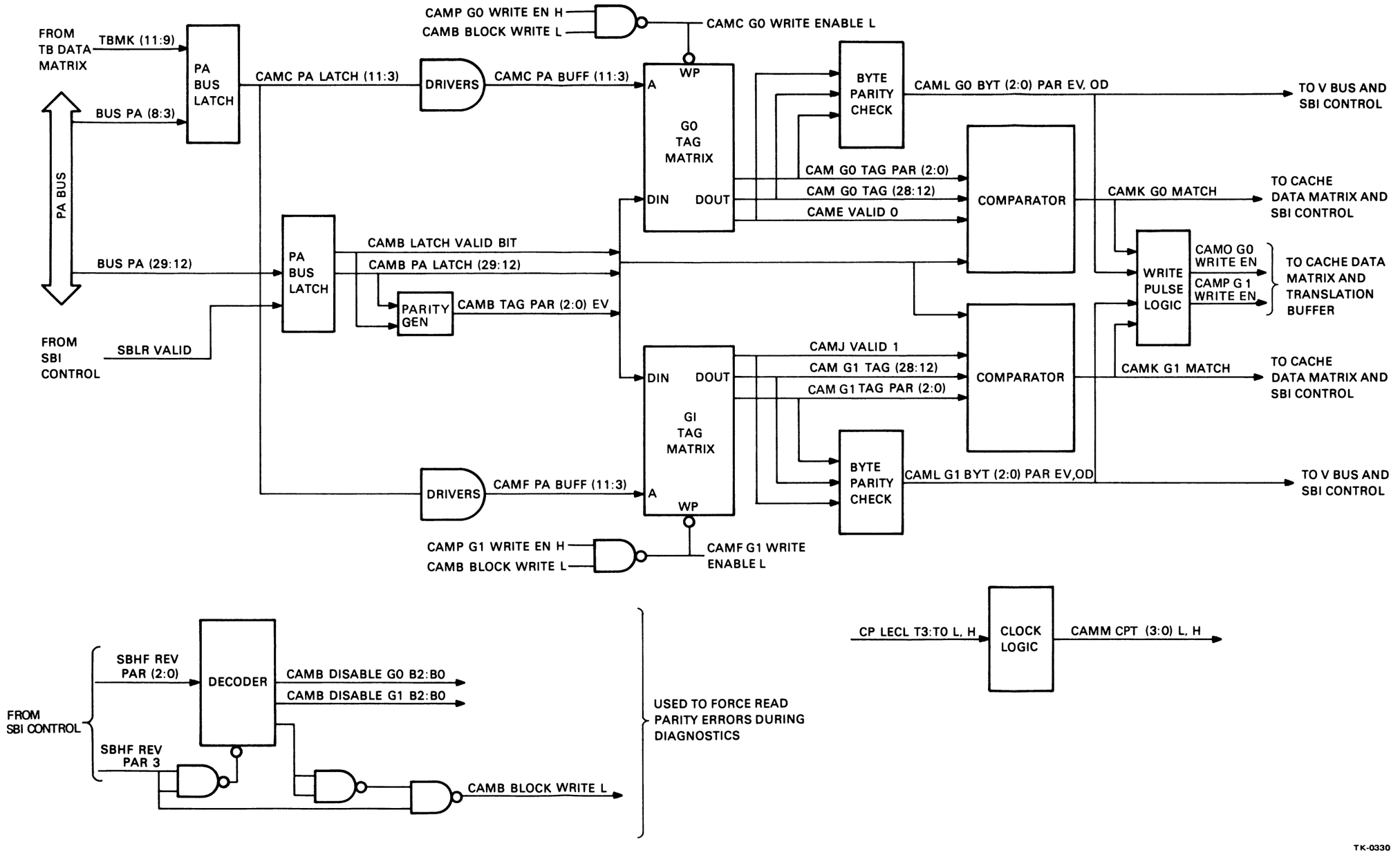


Figure 2-16 Cache Address Matrix

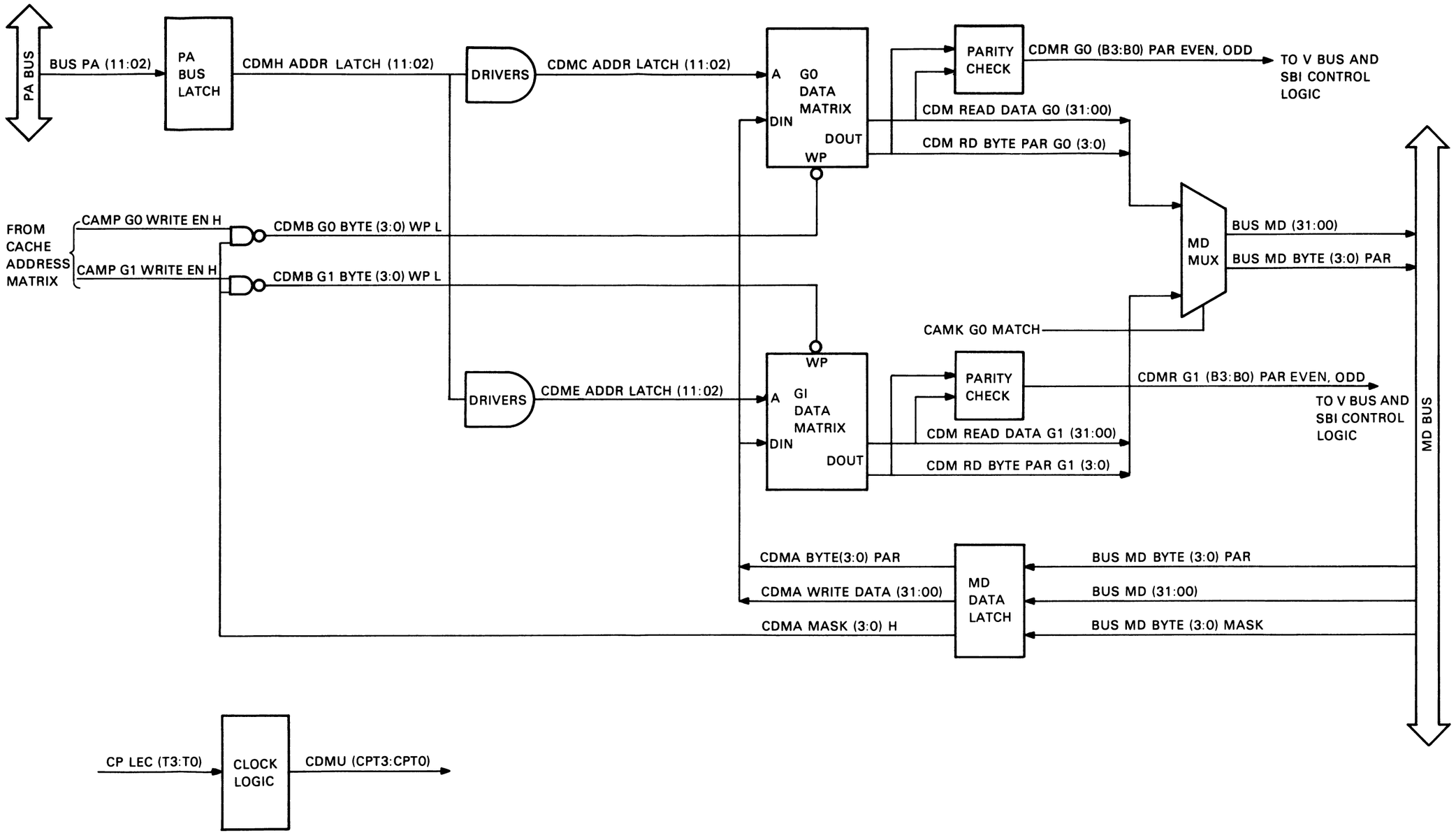


Figure 2-17 Cache Data Matrix

2.2.4.3 Address Parity – Cache generates parity bits for the valid and tag bits which are stored in the address matrix as a result of a read miss. The parity bits [CAMB TAG PAR (2:0)EV] are stored with the corresponding tag and valid bit. When the address matrix is accessed to determine whether a Cache hit or miss occurred, the contents of the indexed location in both groups are parity checked. Detection of a parity error on a read results in a Cache parity microtrap. If a parity error occurs on a write miss, the SBI Control writes arbitrary data with good parity into the location and marks the block invalid by dropping the valid bit.

2.2.4.4 Data Parity – When data is written into Cache, parity bits [CDMA BYTE (3:0)] for the data are latched from the MD bus and stored with the data. Cache checks for correct parity when a location is indexed. Data from the indexed location of both groups is parity checked. If a parity error is detected on a read, a Cache parity microtrap occurs. If a parity error is detected during a write miss, the SBI Control writes arbitrary data with good parity into the location and marks the block invalid by dropping the valid bit.

2.2.4.5 Stall Signal – Whenever a Cache read miss occurs, the requested data must be fetched from main memory by the SBI Control. In accommodation, the SBI Control generates a stall signal (SBLT STALL L) and sends it to the microsequencer to delay CPU operation. This signal temporarily prevents the execution of the next microinstruction until the data is fetched (Paragraph 2.3.2.8).

The CPU is also stalled for all writes by the data paths when the SBI Control write buffer is full. In this case the stall signal is asserted until the buffer becomes available.

2.3 SBI CONTROL DESCRIPTION

The SBI Control interfaces the CPU to the SBI and thus conforms to SBI protocol. A description of SBI protocol is provided in the following paragraphs as an introduction to the operation and logic of the SBI Control.

2.3.1 SBI Protocol

The SBI interconnects the KA780 with the memory system and all adapters in the system. The following subsections describe all interconnect lines and their associated communication protocol.

2.3.1.1 Interconnect Synchronization – Six control group lines are clock signals and are used as a universal time base for all nexus connected to the SBI. All SBI clock signals are generated on the CPU clock module and provide a 200 ns clock period.

The clock signals, in conjunction with the standard nexus clock logic, provide the derived blocks within an attached nexus to synchronize SBI activity. Two clock signals (TPH and TPL) produce the basic nexus time states. The remaining four (PCLKH, PCLKL, PDCLKH, and PDCLKL) are phased clocks and help compensate for the clock distribution skew due to cable, backplane, and driver/receiver propagation delays.

2.3.1.1.1 Derived Time States – The derived clocks (within the nexus) define four, 50 ns (nominal) time states in one clock period. The time states (T0, T1, T2, and T3) determine the transmit and receive times on the SBI, with T0 representing the start of a particular clock period. Figure 2-18 illustrates the phase and timing relationships required to generate the individual derived time states. Note that T0 internal to the CPU (CPT0) is not the same as SBI T0. CPT0 corresponds to SBI T1. All nexus need a minimum of T0 and T2.

2.3.1.1.2 Transmit Data – Immediately prior to T0, a transmitting nexus enables its transmit enable inputs to the SBI transceivers. At T0, the data buffer is clocked and its content enabled to the information path of the SBI. Figure 2-19 is a basic block diagram for one SBI information path line.

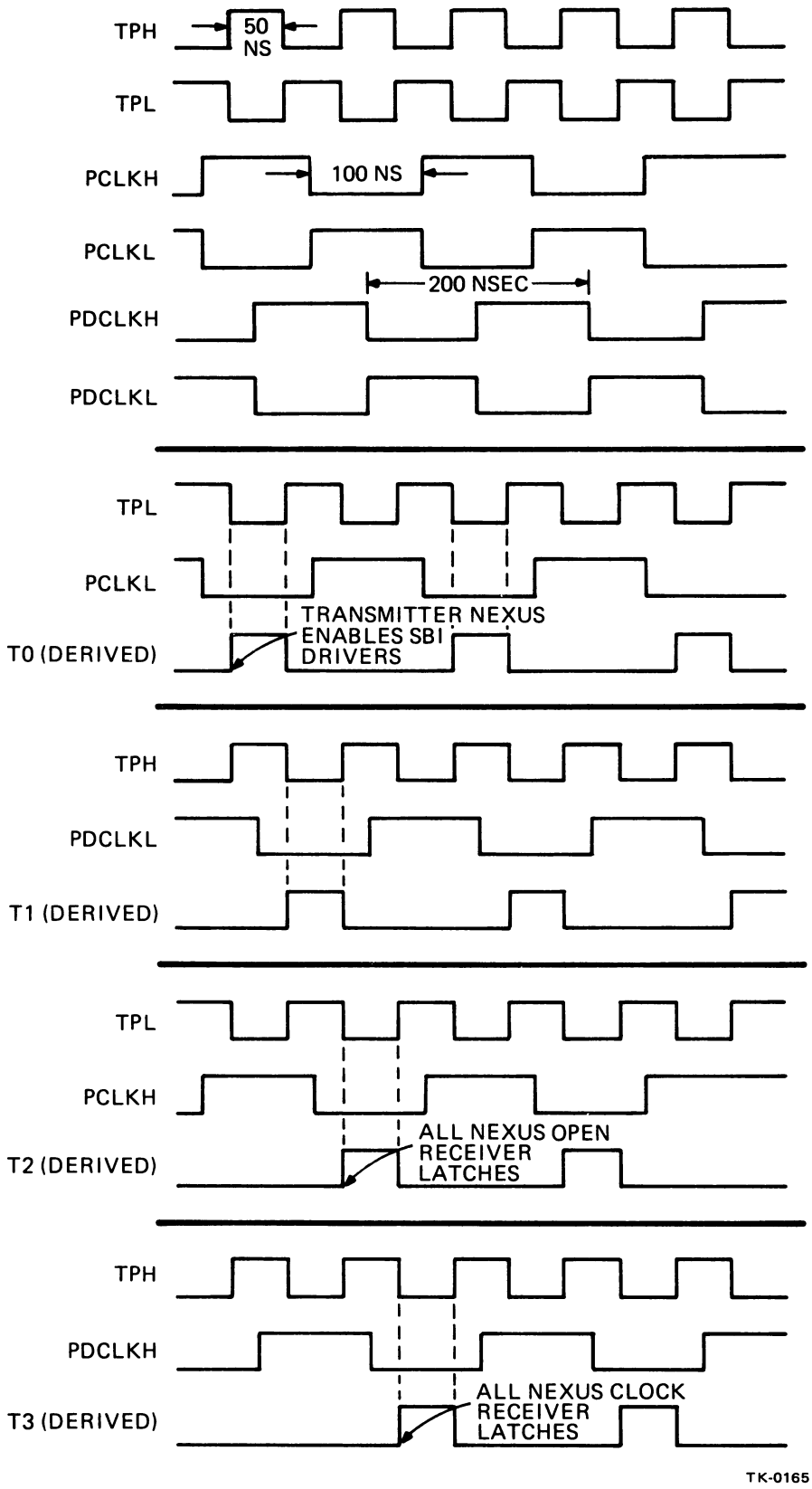


Figure 2-18 SBI Time and Phase Relationships

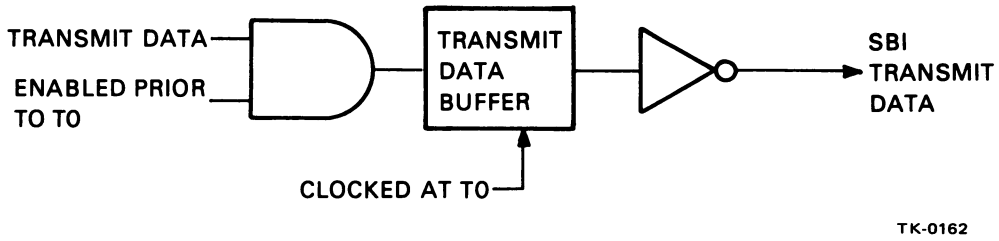


Figure 2-19 Transmit Data Path

2.3.1.1.3 Receive Data – In the case of receive data, nexus receiver latches are opened at T2 and latched at T3. Figure 2-20 shows the basic one-line receiver latch logic. Note that the information may be considered undefined between T2 and T3; only after T3 is information considered valid. Nexus checking, decoding, and subsequent decision making are then based on these latched signals.

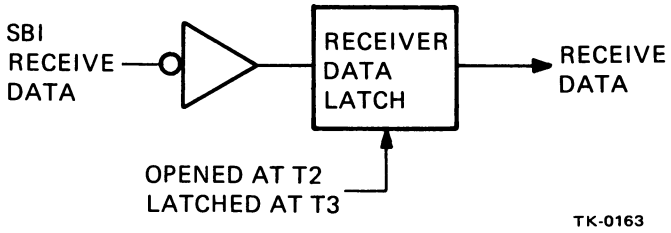


Figure 2-20 Receive Data Path

2.3.1.1.4 Single Time States – In single time states, the time between any T0-T1, T1-T2, T2-T3, and T3-T0 may vary from 50 ns (nominal) to an indefinitely long period of time. SBI operation and protocol will proceed normally. Nexus implement the SBI timeout functions by counting SBI cycles. Memory nexus operation must be normal even though the timing may be different. Nexus that derive timing from an external source (e.g., a mass storage device) set data late and overrun error bits as appropriate. However, the SBI operation of these nexus remains normal.

2.3.1.2 SBI Summary – Table 2-4 summarizes the signal fields associated with each functional group. Figure 2-21 shows the SBI configuration. The following subsections provide detailed descriptions of the individual group field layouts and functions.

Table 2-4 SBI Field Summary

Field	Description
Arbitration Group	
Arbitration Field [TR (15:00)]	Establishes a fixed priority among nexus for access to and control of the information transfer path.
Information Transfer Group	
Information Field [B (31:00)]	Bidirectional lines that transfer data, command/address, and interrupt information between nexus.
Mask Field [M(3:0)]	Primary function: Encoded to indicate a particular byte within the 32-bit information field [B (31:00)].
	Secondary function: In conjunction with the Tag field, indicates a particular type of read data.
Identifier Field [ID (4:0)]	Identifies the logical source or destination of information contained in B(31:00).
Tag Field [TAG (2:0)]	Defines the transmit or receive information types and the interpretation of the content of the ID and information fields.
Function Field [F (3:0)]	Specifies the command code, in conjunction with the Tag field. This field is valid as part of the 32-bit information field only when the Tag equals command/address.
Parity Field [P (1:0)]	Provides even parity for all information transfer path fields. P(1) is generated as parity for the information field. P(0) is generated as parity for the Tag, ID, and mask fields.
Response Group	
Confirmation Field [CNF (1:0)]	Asserted by a receiving nexus to specify one of four response types and indicate its capability to respond to the transmitter request.
Fault Field (FAULT)	A cummulative error line which indicates one of several errors on the SBI.

Table 2-4 SBI Field Summary (Cont)

Field	Description
Interrupt Request Group	
Request Field [REQ (7:4)]	Allows a nexus to request an interrupt to service a condition requiring CPU intervention. Each request line represents a level of nexus request priority.
Alert Field (ALERT)	A cummulative status line which allows those nexus not equipped with an interrupt mechanism to indicate a change in power or operating conditions.
Control Group	
Clock Field (CLOCK)	Six control lines which provide the clock signals necessary to synchronize SBI activity.
Fail Field (FAIL)	A single line from nexus required to initiate a system bootstrap operation.
Dead Field (DEAD)	A single line to the CPU to indicate an impending clock circuit or SBI terminating network power failure.
Unjam Field (UNJAM)	A single line from the CPU to attached nexus which restores the nexus to a known state.
Interlock Field (INTLK)	A single line which provides coordination among nexus responding to certain read/write commands to ensure exclusive access to shared data structures.

2.3.1.3 Arbitration Group Functions and Assignments – The arbitration lines [Transfer Request TR (15:00)] allow up to 16 nexus to arbitrate for the information lines (information transfer group). One arbitration line is assigned to each nexus to establish the fixed priority access. Priority increases from TR15 to TR00, where TR00 is the highest. The lowest priority level is reserved for the CPU, and it requires no actual TR signal line. The other 15 nexus are assigned TR15 through TR01.

The highest priority level, TR00, is reserved as a hold signal for those nexus that require more than one successive SBI cycle. TR00 may only be used by nexus that require:

1. Two or three adjacent cycles for a write-type exchange.
2. Two adjacent cycles for an extended read exchange.
3. Adjacent cycles for interrupt summary read exchanges.

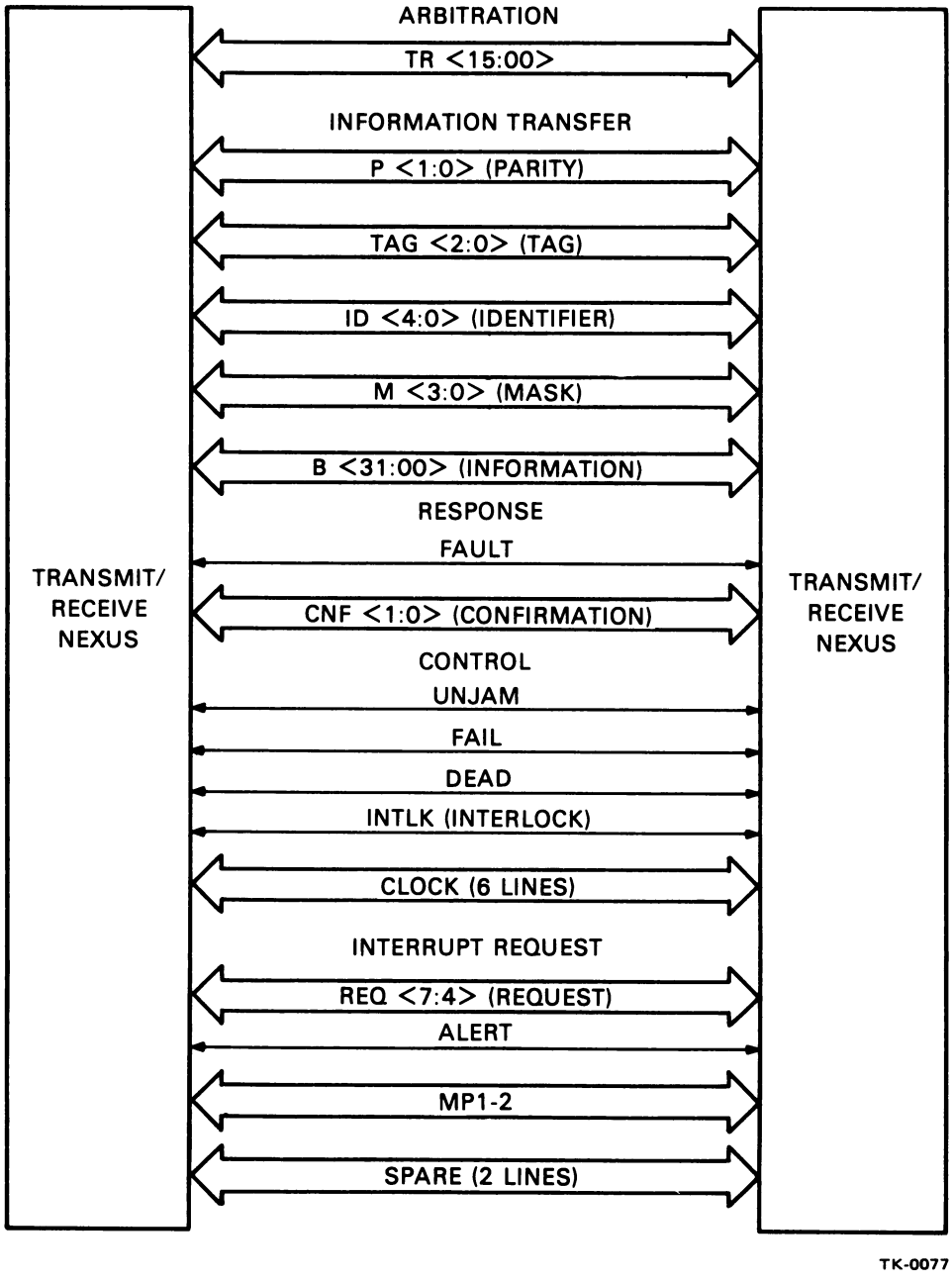


Figure 2-21 SBI Configuration

A nexus requests control of the information path by asserting its assigned TR line at T0 of an SBI cycle. At T3 of the same SBI cycle, the nexus examines (arbitrates) the state of all higher priority TR lines. If no higher priority TR lines (lower TR number) are asserted, the requesting nexus assumes control of the information path at T0 of the following SBI cycle. At this T0 time state, the nexus negates its TR line and asserts command/address or data information on B(31:00). In addition, if a write-type exchange is specified, the nexus asserts TR00 to retain control of adjacent SBI cycles.

If higher priority TR lines are asserted, the requesting nexus can not gain control of the information path. The nexus keeps its TR line asserted and, again, examines the state of higher priority lines at T3 of the next SBI cycle. As before, if no higher TR lines are asserted, the nexus assumes information path control at T0.

2.3.1.4 Information Transfer Group Description – Each information group field is described in detail in the following subsections.

2.3.1.4.1 Parity Field – The parity field [P(1:0)] provides even parity for detecting single bit errors in the information group (Figure 2-22).

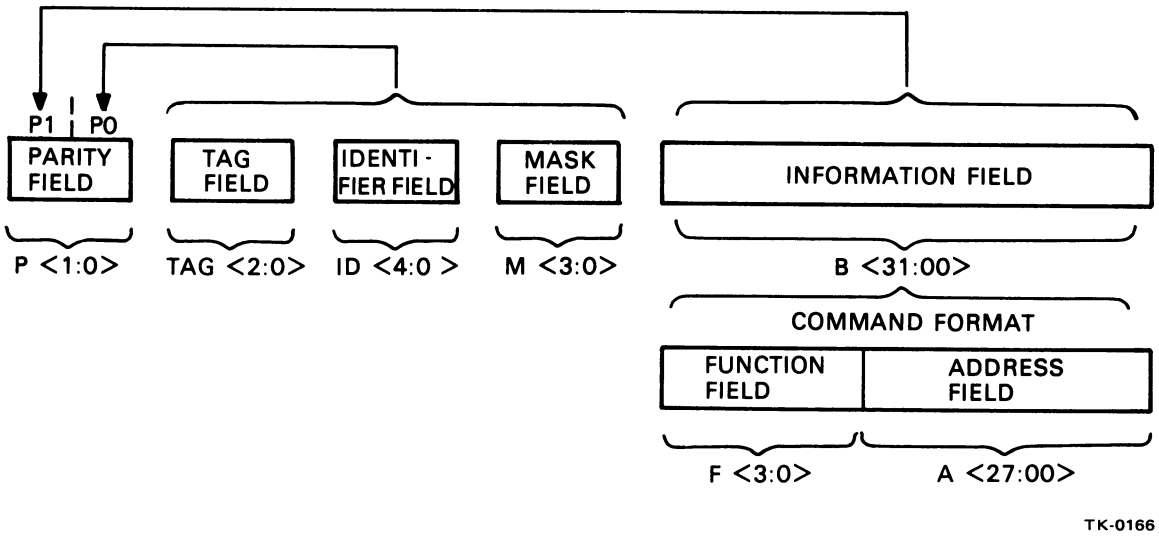


Figure 2-22 Parity Field Configuration

A transmitting nexus generates P0 as parity for TAG (2:0), ID (4:0), and M(3:0). The P1 parity bit is generated for B(31:00). P0 and P1 are generated so that the sum of all logic one bits in the checked field, including the parity bit, is even. With no SBI transmissions, the information transfer path assumes an all zeros state; thus, P(1:0) should always carry even parity. Any transmission with odd parity is considered an error.

2.3.1.4.2 Tag Field – The Tag field [TAG (2:0)] is asserted by a transmitting nexus to indicate the information type being transmitted on the information lines. The tag field determines the contents of the B field. The following subsections describe each information type, tag code, and associated field content.

Command/Address Tag – A tag field content of 011 indicates that the content of B(31:00) is a command/address word. ID(4:0) asserted at this time is a unique code identifying the logical source (commander) of the command. As shown in Figure 2-23, B(31:00) is divided into a function field and an address field to specify the command and its associated address.

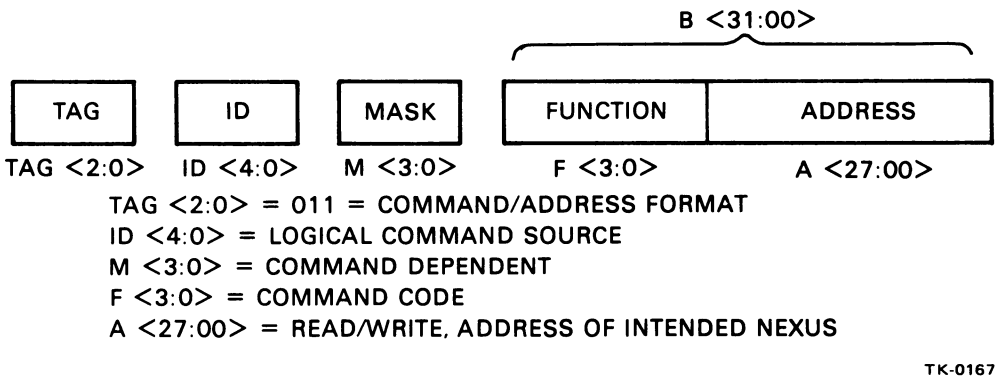


Figure 2-23 Command/Address Format

In a write-type command, the ID field code represents the logical source and the address field specifies the logical command destination. For a read-type command, the addressed nexus holds the transmitted ID for transmission with the requested data. The ID is sent with the read data to indicate destination.

The 28 bits of the SBI address field define a 268, 435, 456 long word address space, which is divided into two sections. Addresses 0-7FFFFFF$_{16}$ are reserved for primary memory. Addresses 8000000$_{16}$ and -FFFFFFF$_{16}$ are reserved for device control registers. Generally, primary memory begins at address 0; the address space is dense and consists only of storage elements. The control address space is sparse with address assignments based on device type. Each nexus is assigned at 2048, 32-bit longword address space for control. The addresses assigned are determined by the TR number as shown in Figure 2-24.

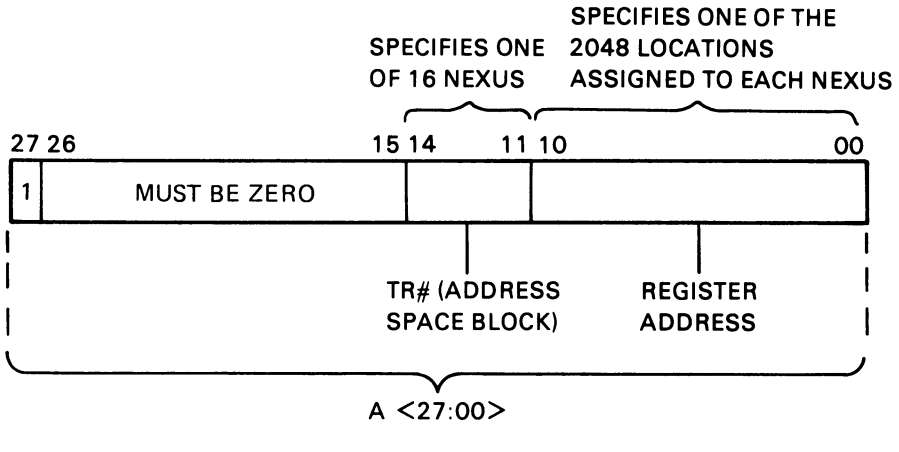


Figure 2-24 Control Address Space Assignment

Read Data Tag – A tag field content of 000 indicates that B(31:00) contains data requested by a previous read-type command. In this case, ID(4:0) is a unique code which was received with the read command and identifies the logical destination of the requested data. The retrieved data may be one of three types: read data, corrected read data, and read data substitute where the particular type is identified by M(3:0).

Read data is the normally expected error-free data having M(3:0) = 0000. See Figure 2-25.

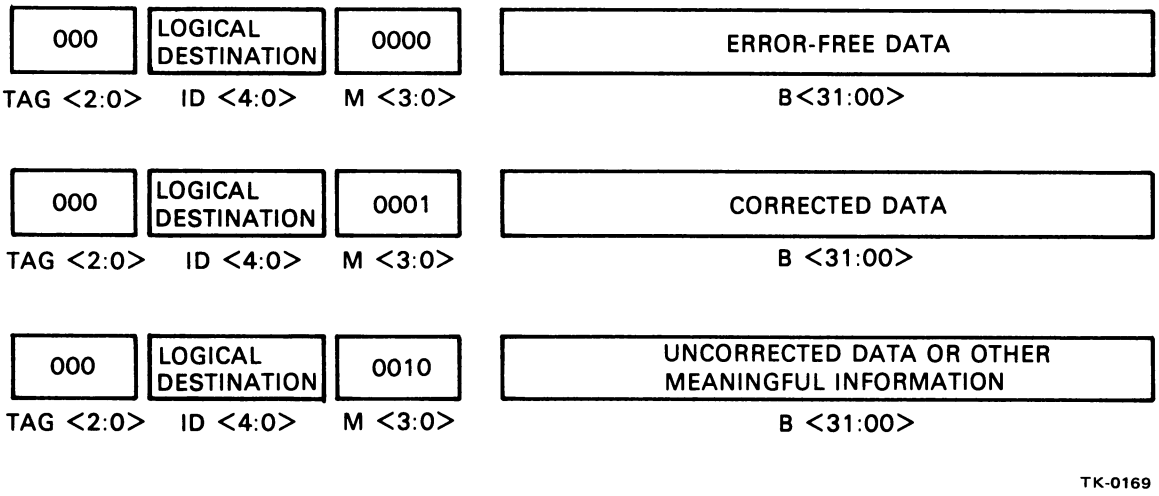


Figure 2-25 Read Data Formats

Corrected read data is data in which an error was detected and subsequently corrected by the error correction code logic (ECC) of the device transmitting the read data. In this case, the mask field flags the corrected data with M(3:0) = 0001.

Read data substitute represents data in which an error was detected but could not be corrected. In this case, B(31:00) will contain the substitute data in the form of uncorrected data or other meaningful information. The mask field flags the uncorrected data with M(3:0) = 0010. As with the other read data types, the ID field identifies the read commander.

Write Data Tag – A tag field content of 101 indicates that B(1:00) contains the write data for the location specified in the address field of the previous write command (Figure 2-26). The write data will be asserted on B(31:00) in the SBI cycle immediately following the command/address cycle. The mask field specifies bytes within B(31:00) for the operation.

Interrupt Summary Tag – A tag field content of 110 defines B(31:00) as the interrupt level mask for an interrupt summary read command. The level mask [B(07:04)] is used to indicate the interrupt level being serviced as the result of an interrupt request. In this case, the ID field identifies the commander, which is usually a CPU. Although unused, M(3:0) must be transmitted as zero.

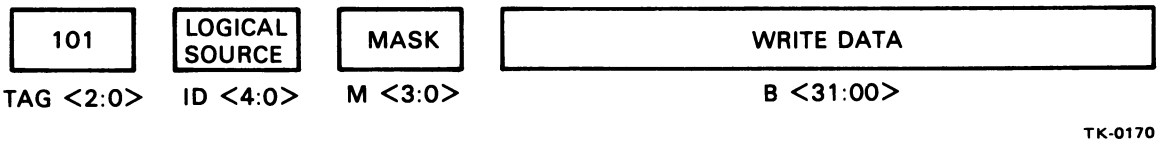


Figure 2-26 Write Data Format

The interrupt sequence consists of two exchanges:

1. The first exchange indicates the interrupt level being serviced.
2. The second exchange is the response, where the device requesting the interrupt identifies itself.

The interrupt summary read and response formats are illustrated in Figure 2-27. Note that the interrupt summary response encodes TAG (2:0) = 000.

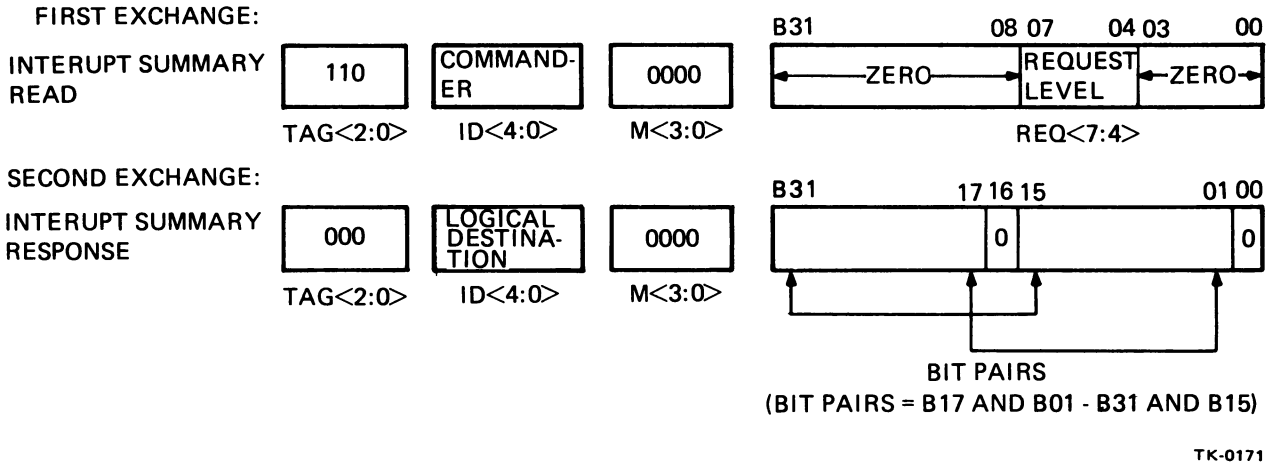


Figure 2-27 Interrupt Summary Formats

Reserved Tag Codes – Tag Code 111 is reserved for diagnostic purposes. Tag codes 001, 010, and 100 are unused and reserved for future definition.

2.3.1.4.3 Identifier Field – The ID field [ID(4:0)] contains a code which identifies the logical source or logical destination of the information contained in B(31:00). Each nexus is assigned an ID code which corresponds to the TR line which it operates. For example, a nexus assigned TR05 would also be assigned ID code = 5. More than one ID code may be assigned to a nexus.

Nexus using more than one code take the first code from the standard ID code assignment (0–15). The second code is taken from the range 17–30 (i.e., first ID code + 16).

Certain ID codes are reserved: ID = 16, unit processors; ID = 31, diagnostic purposes. ID = 0 is reserved so that the idle state of the SBI (read data, destination ID = 0) will not cause a nexus selection. Note that even though a nexus is not selected, all nexus are checking for correct SBI parity.

2.3.1.4.4 Mask Field - The mask field [M(3:0)] has two interpretations. For the first interpetation, M(3:0) is encoded to specify particular data bytes of an addressed location for an operation. This mask interpretation is used with the masks of Read Masked, Write Masked, Interlock Read Masked, Interlock Write Masked, and Extended Write Masked commands and also with masks of Write Data formats. As shown in Figure 2-28, each bit in the mask field corresponds to a particular byte on B(31:00).

The second interpretation of the mask field is used when TAG (2:0) = 000 (Read Data). This interpretation defines the data types as specified in Table 2-5. All other mask field codes (0011–1111) are reserved and are interpreted as Read Data Substitute by receiving nexus.

2.3.1.5 Response Group Description - The three response lines are divided into two fields: Confirmation [CNF(1:0)] and Fault [FAULT]. CNF(1:0) informs the transmitter whether or not the information was correctly received or if the receiver can process the command. FAULT is a cumulative error indication of protocol or information path malfunction and is asserted with the same timing as the confirmation field.

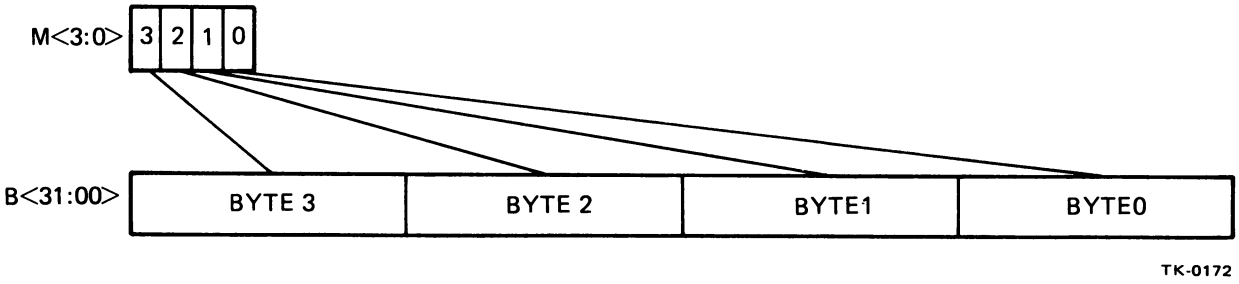


Figure 2-28 Mask Field Format

Table 2-5 Read Data Types

M (3:0)	Data Type
0000	Read Data
0001	Corrected Read Data
0010	Read Data Substitute

Either field is transmitted two cycles after each information transfer. Confirmation is delayed to allow the information path signals to propagate, be checked, and be decoded by all receivers and to be generated by the responder. During each cycle, every nexus in the system receives, latches, and makes decisions on the information transfer signals. Except for multiple bit transmission errors or nexus malfunction, one (or more) of the nexus receiving the information path signals will recognize an address or ID code. This nexus then asserts the appropriate response in CNF.

Any (or all) nexus may assert FAULT after detecting a protocol or information path failure. However, a nexus asserting FAULT may not assert CNF (1:0).

2.3.1.5.1 Confirmation Codes – Table 2-6 lists the confirmation codes and their interpretation.

A BSY (10) or ERR (11) response to transfers other than command/address transfers will be considered as no response from the responder.

Table 2-6 Confirmation Code Definitions

CNF Code	Definitions
00, No Response (N/R)	The unasserted state; it indicates no response to a commander selection.
01, Acknowledge (ACK)	The positive acknowledgment to any transfer.
10, Busy (BSY)	The response to a command/address transfer which indicates successful selection of a nexus which is presently unable to execute the command.
11, Error (ERR)	The response to a command/address transfer which indicates selection of a nexus which cannot execute the command.

2.3.1.5.2 Response Handling – The transmitting nexus samples the CNF and FAULT lines at T3 of the third cycle following transmission. ACK is the expected confirmation response (i.e., command will be executed, or information has been correctly received).

Should a command/address transfer receive a BSY confirmation, the commander must repeat the transmission (after a nominal waiting period) until it is accepted or a timeout occurs.

An N/R confirmation should be treated the same as BSY except that its occurrence may be flagged in a status bit. ERR confirmation is the result of a programming error and should abort the command and invoke the appropriate recovery routine.

Some nexus may be unable to determine within two SBI cycles whether a function will be successfully completed. For these cases, the nexus presumes success and responds with ACK confirmation. If it is later determined that a read-type function cannot be completed, a read data transfer of all zeros is transmitted and an interrupt requested. If a write-type request cannot be completed, the command is aborted and an interrupt requested. In either case, the cause of the interrupt is indicated in a Configuration or Status register.

2.3.1.5.3 Successive Cycle Confirmation – Since Write Masked, Extended Write Masked, and Extended Read operations consist of successive transfers, acknowledgment is more complex.

1. If the command/address transfer is confirmed with N/R or BSY, then no notice will be taken of the data transfer confirmation and the entire sequence will be repeated.

2. If the command/address transfer receives ERR, the sequence is aborted and recovery routines are invoked.

3. If ACK is not received as confirmation for a Write Data command, the command is repeated.

4. Transmissions of read data are confirmed with ACK by the receiver of that data. The read data transmitter may ignore this confirmation, since only commanders execute retry sequences.

2.3.1.5.4 SBI Sequence Timeouts – All commanders implement two timeout functions: Interface sequence timeout and read data timeout. Both timeouts are specified as 102.4 μs (or 512 SBI cycles).

The interface sequence timeout determines the maximum time allowed to complete an interface sequence. The sequence interval is defined as the time from:

1. When SBI arbitration is initiated, until ACK is received for a command/address transfer that specifies read, or

2. When SBI arbitration is initiated, until ACK is received for a command/address transfer that specifies write and ACK is also received for each transmission of write data, or

3. When SBI arbitration is initiated, and an ERR confirmation is received for any command/address transfer.

The read data timeout is defined as the time from when an interface sequence that specified a read command is completed to the time that the specifies read data is returned to the commander. In the case of an Extended Read function both longwords must be retrieved prior to timeout (102.4 μs).

If the last command/address transfer prior to an interface sequence timeout receives an N/R confirmation, it is recorded in a status bit. Certain nexus may terminate their requests for SBI control due to an unusual occurrence in those nexus. When this occurs, both timeouts are cancelled (e.g., when a nexus detects a data late error).

When a timeout occurs, the commander provides the actual address or reconstructed address for which the timeout occurred. In addition, the commander records the type of timeout received (i.e., interface sequence or read data). Either timeout will terminate a command transmission retry.

2.3.1.5.5 Fault Detection – Each nexus is equipped with a 32-bit Configuration and Fault Status register (register 0). The fault status portion of this register contains flags which cause the assertion of the FAULT line. The fault status portion is described in Figure 2-29.

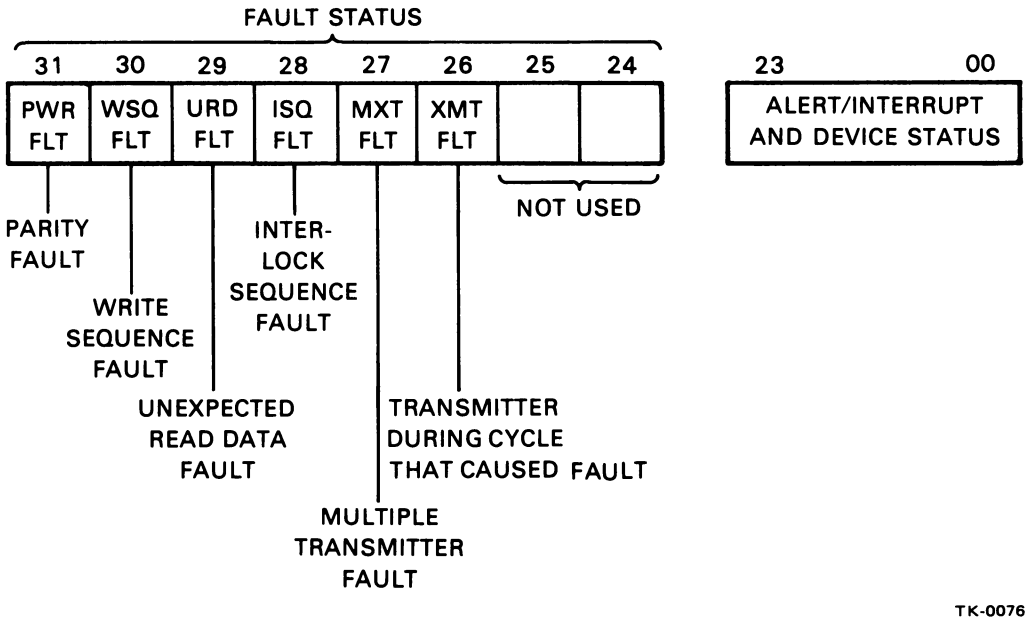


Figure 2-29 Fault Status Flags

A nexus detecting one of the fault conditions will assert the FAULT signal for one cycle. FAULT then causes each nexus on the system to latch its fault status. The fault status bits thus latched refer to the cycle during which the fault occurred. The CPU examines the FAULT signal and latches the signal on the leading edge of FAULT. The CPU then continues to assert FAULT until the software has examined the fault status bits of all nexus and has specified the negation of FAULT. Figure 2-30 shows the timing involved.

Figure 2-31 illustrates the confirmation and fault decision flow for all responses and error conditions.

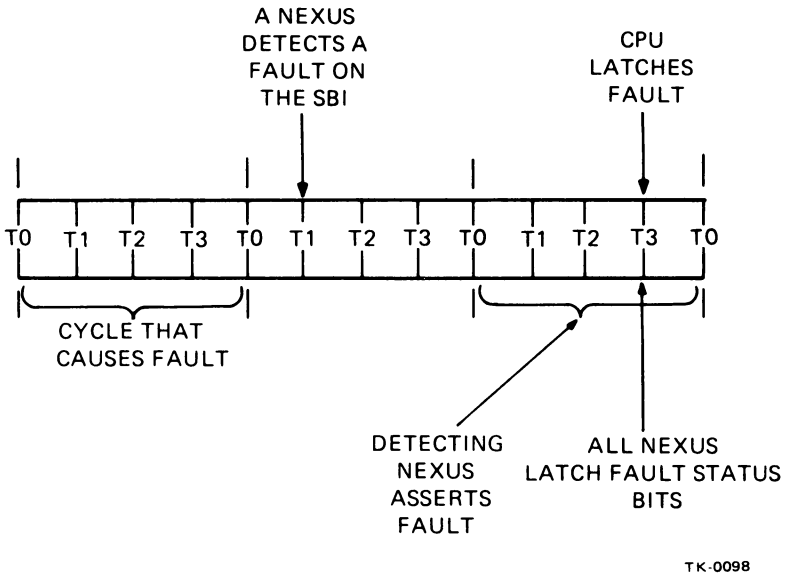
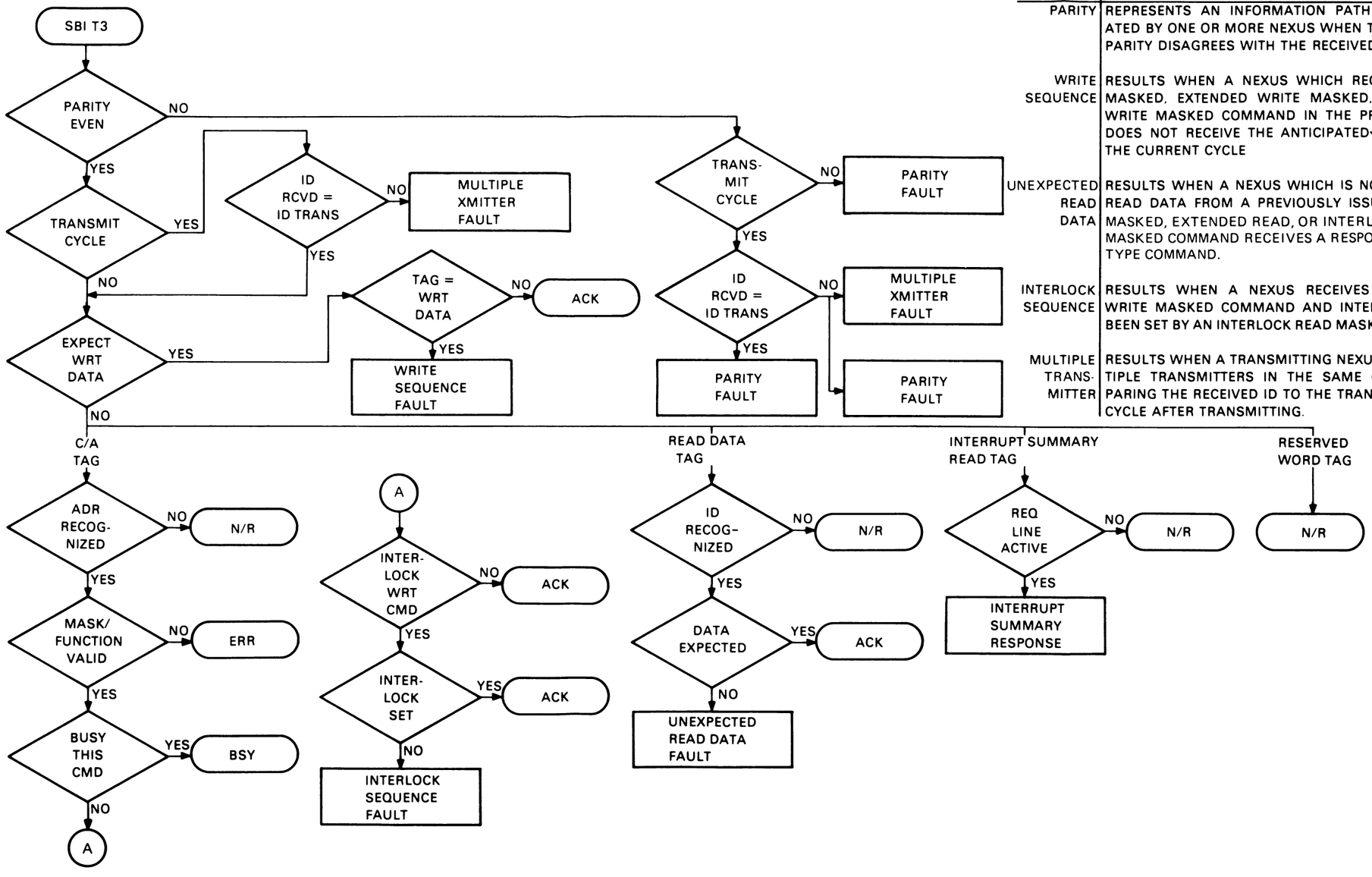


Figure 2-30 Fault Timing



SBI FAULT DEFINITIONS

FAULT	DEFINITION
PARITY	REPRESENTS AN INFORMATION PATH ERROR GENERATED BY ONE OR MORE NEXUS WHEN THE CALCULATED PARITY DISAGREES WITH THE RECEIVED PARITY.
WRITE SEQUENCE	RESULTS WHEN A NEXUS WHICH RECEIVED A WRITE MASKED, EXTENDED WRITE MASKED, OR INTERLOCK WRITE MASKED COMMAND IN THE PRECEDING CYCLE DOES NOT RECEIVE THE ANTICIPATED WRITE DATA IN THE CURRENT CYCLE
UNEXPECTED READ DATA	RESULTS WHEN A NEXUS WHICH IS NOT WAITING FOR READ DATA FROM A PREVIOUSLY ISSUED READ MASKED, EXTENDED READ, OR INTERLOCK READ MASKED COMMAND RECEIVES A RESPONSE TO A READ TYPE COMMAND.
INTERLOCK SEQUENCE	RESULTS WHEN A NEXUS RECEIVES AN INTERLOCK WRITE MASKED COMMAND AND INTERLOCK HAS NOT BEEN SET BY AN INTERLOCK READ MASKED COMMAND.
MULTIPLE TRANS-MITTER	RESULTS WHEN A TRANSMITTING NEXUS DETECTS MULTIPLE TRANSMITTERS IN THE SAME CYCLE BY COMPARING THE RECEIVED ID TO THE TRANSMITTED ID ONE CYCLE AFTER TRANSMITTING.

TK-0086

Figure 2-31 Confirmation and Fault Decision Flow

2.3.1.6 Interrupt Request Group Description – The interrupt request group consists of four request lines [REQ (7:4)] and an alert [ALERT] line. Request lines are assigned to some of the nexus and represent assigned CPU interrupt levels. The lines are used by nexus to invoke a CPU to service a condition requiring processor intervention. The request lines are priority encoded in an ascending order of REQ4-REQ7. A requesting nexus asserts its request lines (or line) synchronously with respect to the SBI clock to request an interrupt. Any of the REQ lines may be asserted simultaneously by more than one nexus, and any combination of REQ lines may be asserted by the collection of requesting nexus.

The ALERT signal is asserted by nexus which do not implement interrupt request lines. Its purpose is to indicate to the CPU a change in the nexus power condition or operating environment. Nexus which implement the REQ lines report such changes by requesting an interrupt.

2.3.1.6.1 Interrupt Operation – When a nexus requires an interrupt, it asserts its REQ line on the SBI. At a time judged appropriate, the CPU will recognize the interrupt request and issue an Interrupt Summary Read command [TAG (2:0) = 110]. The command will have a single bit set in its interrupt level mask [B (7:4)] which corresponds to the REQ line being serviced. For example, B 04 set to a logic one indicates that the REQ 4 level is being serviced. Note that the remaining information path fields [i.e., B(31:08), ID (03:00), and M(3:0)] are transmitted as zero.

Nexus receiving the Interrupt Summary Read command without error and asserting the REQ line specified in the interrupt level mask will assert a 2-bit code in B(31:00). This code, which identifies the requesting nexus, is asserted with the timing of CNF (1:0). However, the responding nexus does not assert any CNF, TR, ID, or TAG line; nexus that detect incorrect parity will assert FAULT.

As shown in Figure 2-32, the asserted bits are in corresponding positions in the upper and lower 16 bits of B(31:00). The bit pair uniquely identifies the nexus among those using the particular REQ line. Only 15 bit pairs in the information field are used (i.e., B31 and B15 through B17 and B01). Since only pairs of bits are asserted, parity remains correct regardless of the number of responding nexus. The two bits asserted by the requesting nexus are equal to the nexus TR number and the nexus TR number plus 16.

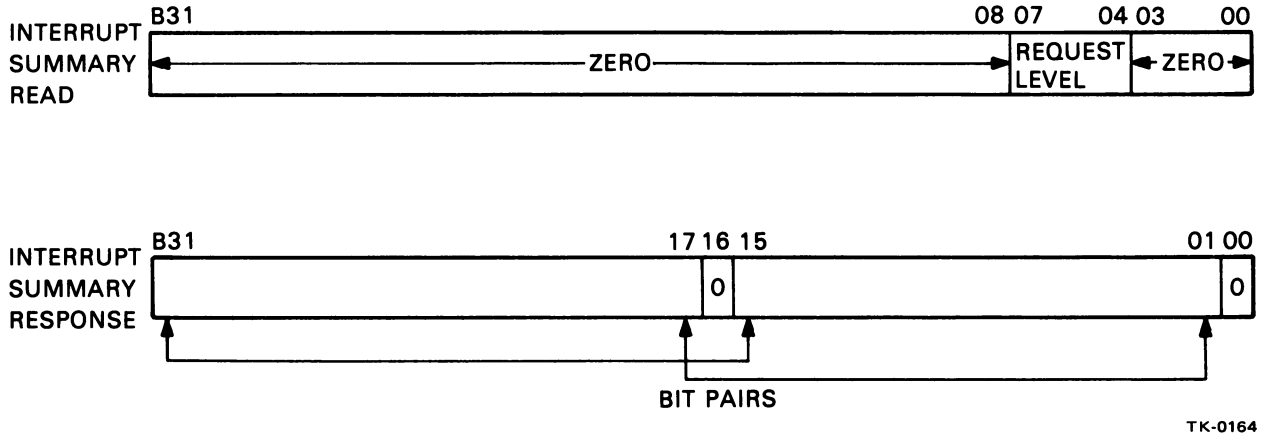


Figure 2-32 Request Level and Nexus Identification

While holding control of the SBI with TR00, the CPU waits two cycles after the Interrupt Summary Read command is transmitted before latching B(31:00) into an internal register. By encoding the REQ level and the bit pair received from responding nexus, the CPU generates a vector unique to that level and nexus. The vector in turn is used to invoke the nexus service routine. The service routine will take explicit action by writing a device register to clear the interrupt condition. Clearing the interrupt causes the nexus to negate the REQ line, provided the nexus does not have any other outstanding interrupts at this level. The negation of REQ occurs within two cycles of the write data transmission.

Normally, the CPU will service requests in the descending order REQ7 through REQ4. Similarly, nexus are identified in descending order beginning with the nexus which asserts bits B31 and B15 and ending with the nexus which asserts bits B17 and B1. If mulitple nexus are requesting interrupts on the same REQ line, multiple Interrupt Summary Read commands are issued until all nexus have been serviced and the REQ line is no longer asserted.

Figure 2-33 is a functional timing chart for the interrupt operation.

2.3.1.6.2 Status Register Alert Flags - As shown in Figure 2-34, each nexus maintains bits in its Configuration register to indicate conditions which cause assertion of ALERT (or the appropriate REQ line if implemented). Power down and power up status bits are provided, but additional ALERT status bits are present if other conditions such as over-temperature are detectable.

The ALERT line is the logical OR of the ALERT status bits and is asserted synchronously to the SBI clock. ALERT status bits are cleared when written as logic one; when written as logic zero, they are not changed. These status bits are also cleared when the UNJAM signal is received.

2.3.1.6.3 Alert Flag Operation - A nexus asserts ALERT or an interrupt request when any of its ALERT status bits are set. The bits are set during the following events:

1. During power failure at the nexus when the assertion of power supply AC LO is recognized.

2. During the restoration of power when the negation of AC LO is recognized.

3. When other environmental conditions such as over-temperature are detected.

The alert status bits are only set on the transition of the event that caused them to set.

The power down status bit is set when there is a transition of the nexus AC LO from the negated to the asserted state. Setting the power down status bit clears the power up status bit; likewise, setting the power up bit clears the power down bit. The over-temperature bit is set when there is a transition from the normal to the over-temperature state.

A nexus asserting ALERT or asserting an interrupt request due to an Alert status bit set, continues to assert ALERT until:

1. All ALERT status bits are cleared (written with a logic on).

2. UNJAM signal is received.

3. Nexus loses dc power.

The negation of ALERT (or REQ) is synchronous to the SBI clock and occurs within two cycles of the write data transmission used to clear the ALERT condition.

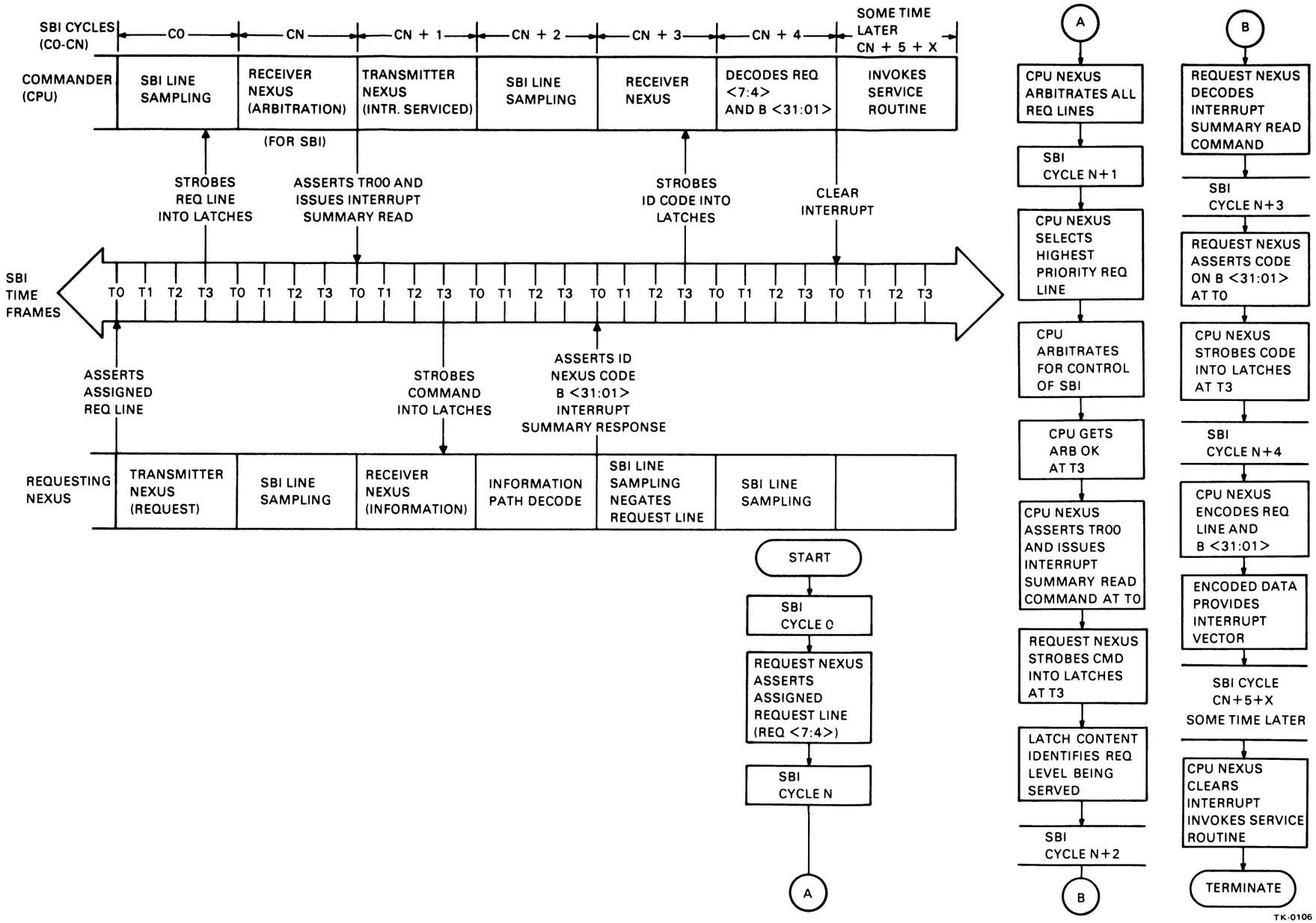


Figure 2-33 Interrupt Operation Timing and Flow

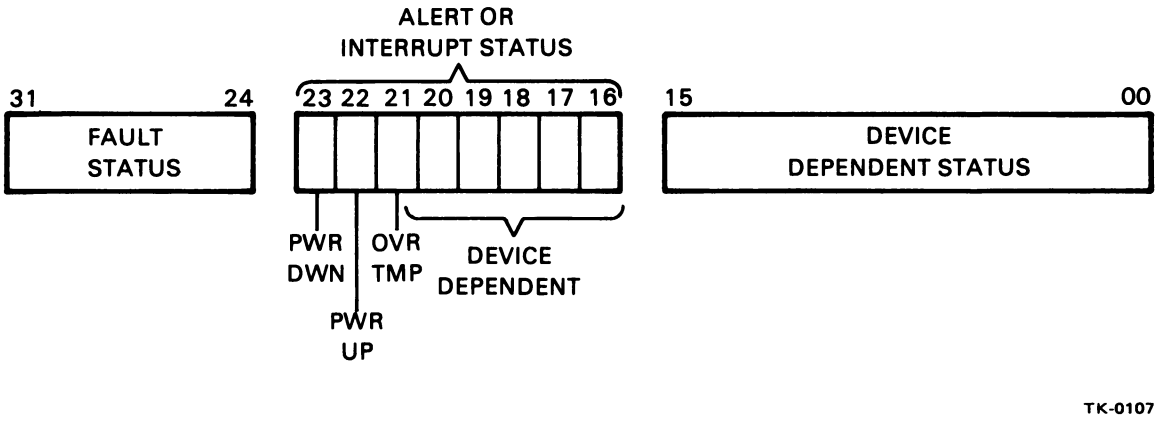
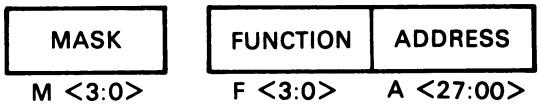


Figure 2-34 Alert Status Bits

2.3.1.7 Command Code Description – The operations executed over the SBI are specified in command/address (C/A) format using the mask, function, and address fields. Figure 2-35 summarizes the command/address formats and lists the command codes. Several function codes are unused and reserved for future use. All nexus must respond to these reserved codes with an N/R confirmation.



MASK USE	FUNCTION CODE	FUNCTION DEFINITION
IGNORED	0000	RESERVED
USED	0001	READ MASKED
USED	0010	WRITE MASKED
IGNORED	0011	RESERVED
USED	0100	INTERLOCK READ MASKED
IGNORED	0101	RESERVED
IGNORED	0110	RESERVED
USED	0111	INTERLOCK WRITE MASKED
IGNORED	1000	EXTENDED READ
IGNORED	1001	RESERVED
IGNORED	1010	RESERVED
USED	1011	EXTENDED WRITE MASKED
IGNORED	1100	RESERVED
IGNORED	1101	RESERVED
IGNORED	1110	RESERVED
IGNORED	1111	RESERVED

TK-0083

Figure 2-35 SBI Command Codes

2.3.1.7.1 Read Masked Function – The read masked function is specified in Figure 2-36.

Prior to issuing the command, the commander asserts its TR line to arbitrate for SBI Control. When the commander gains control of the SBI, it asserts the information transfer lines at T0. At T3 of the same cycle, the receiver nexus strobes the command/address information into its receiver latches for decoding. The C/A format presented on the SBI instructs the nexus selected by the address field, A (27:00), to retrieve the data addressed by A (27:00) and the mask and transfer it to the logical destination specified in the ID field. The addressed nexus will respond to the C/A transfer with ACK (assuming no errors), two or more SBI cycles after the assertion of C/A.

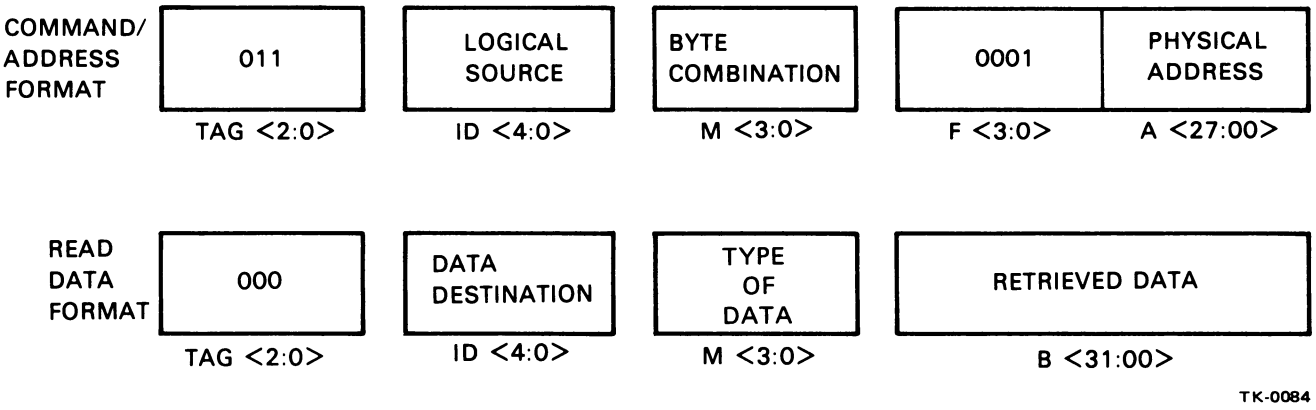


Figure 2-36 Read Masked Function Format

The addressed data is retrieved in a time frame which is dependent on the nexus response time. Following the response delay, the responding nexus must arbitrate for control of the SBI. After ARB OK is true on the responder, the information fields are asserted on the SBI to T0. TAG (2:0) is coded as 000 specifying the read data format and ID (4:0) is coded to identify the logical destination. The read data is asserted on B(31:00) and transferred to its destination as read data [M(3:0) = 000] or as corrected read data [M(3:0) = 0001]. In the case of uncorrectable read data, the addressed nexus transmits read data substitute [M(3:0) = 0010].

After the assertion of read data, the commander latches the content of B(31:00) at T3 of the same SBI cycle. At T0 two cycles later, the commander confirms the successful transfer by asserting ACK.

Figure 2-37 is a functional timing chart for the read masked operation.

2.3.1.7.2 Extended Read Function – The Extended Read function is similar to the Read Masked function in operation. The function format is shown in Figure 2-38.

The mask field and bit A00 of the received command/address word are ignored. However, the mask field must be transmitted as zero.

In the Extended Read function, 64 bits (two data longwords) are always transmitted and thus require two SBI data transfer cycles. In this case, F(3:0) instructs the nexus selected by A(27:01) to retrieve the addressed 64-bit data and transfer it to the logical destination (specified in the ID field) as in the Read Masked function. The first transmission transfers the even longword (A00 = 0), and the second transfers the odd longword (A00 = 1).

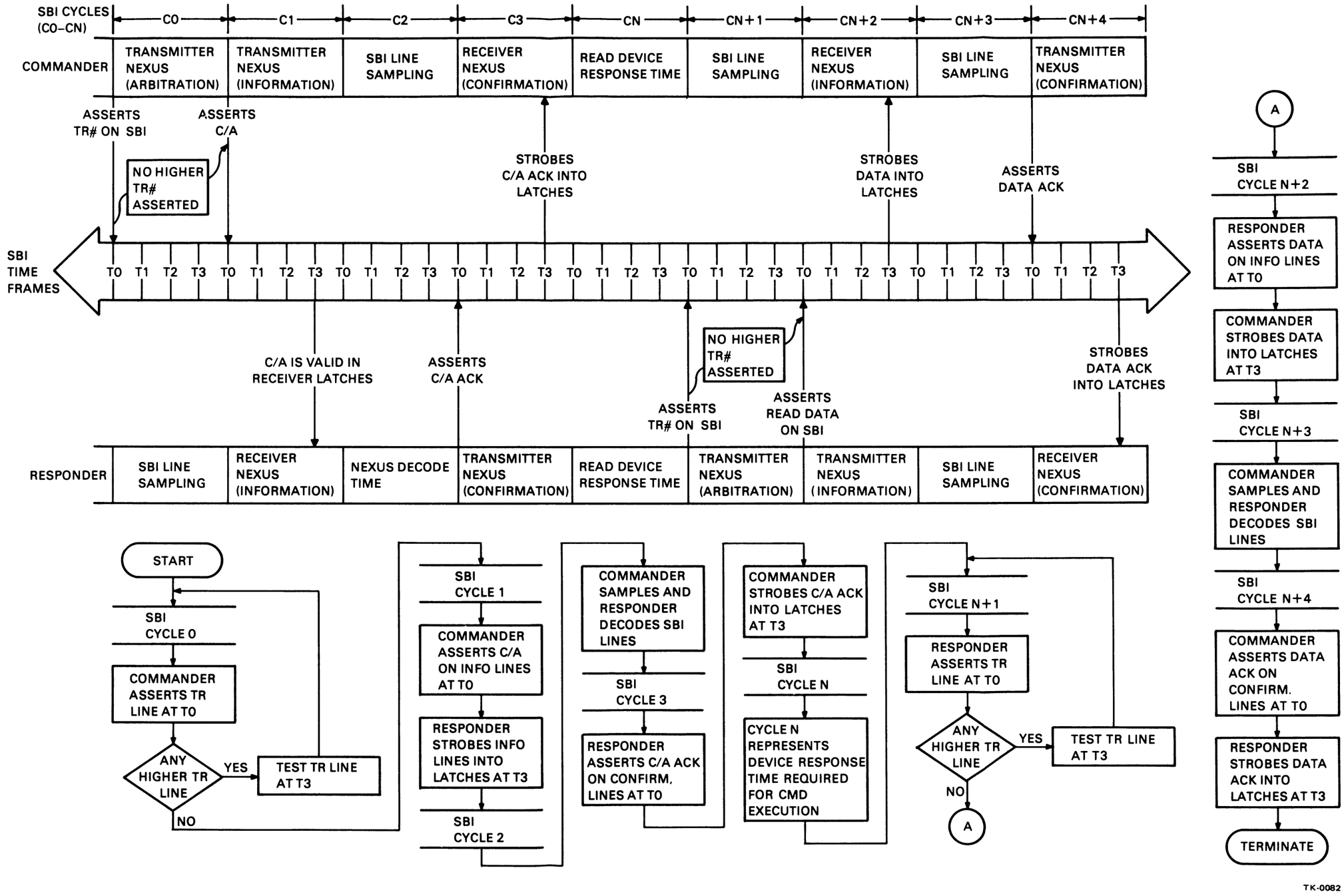


Figure 2-37 Read Masked Timing and Flow

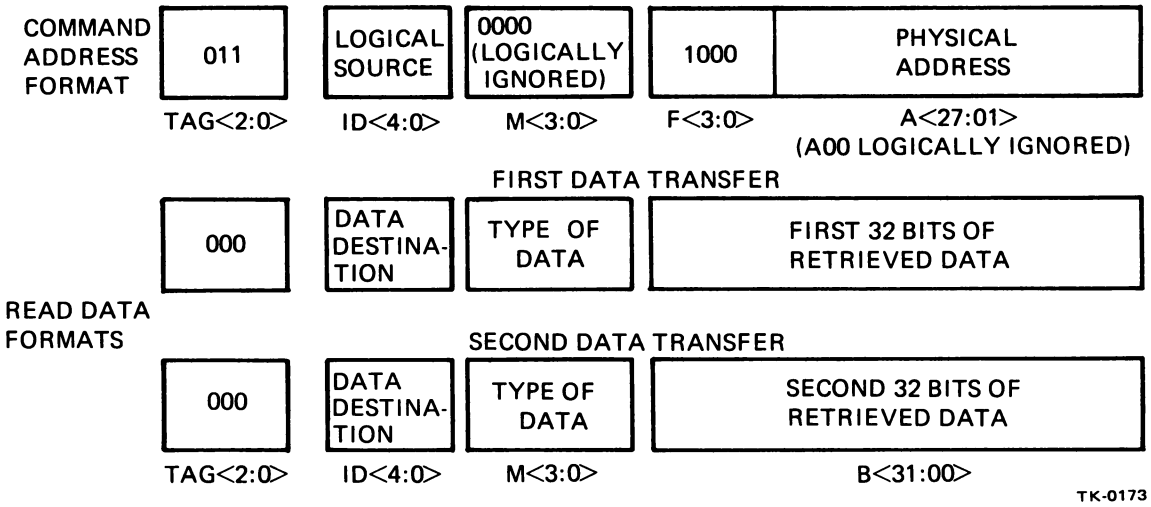


Figure 2-38 Extended Read Function Format

When the commander gains control of the SBI, it asserts the C/A information at T0. At T3 of the same cycle, the receiver nexus strobes the C/A information into its receiver latches for decoding. The addressed nexus confirms the C/A transfer by returning ACK two cycles after the assertion of C/A. Following the response delay and arbitration, the responder asserts the first 32-bit data longword (A00 = 0) on B(31:00). The other information fields are coded as in the Read Masked operation. The second data longword (A00 = 1) is asserted on B(31:00) at T0 of the succeeding cycle. The mask field describing the data type will be asserted with each read data longword.

The commander latches B(31:00) (first data word) at T3 of the cycle when it was transmitted. At T3 of the next cycle, the commander again latches B(31:00) (second data word). Then at T0 of the following cycle, the commander confirms the first data transfer with ACK. The commander confirms the second data transfer with ACK at T0 of the cycle after that.

Figure 2-39 is a functional timing chart showing the Extended Read operation.

2.3.1.7.3 Write Masked Function – The write masked function format is shown in Figure 2-40. F(3:0) instructs the selected nexus to modify the bytes specified by M(3:0) in that storage element addressed by A(27:00) using data transmitted in the succeeding cycle.

When the commander gains control of the SBI, it asserts the C/A information at T0. The mask selects the bytes to be written. The commander also asserts TR00 at T0 to retain control during the succeeding SBI cycle. At T3 of the same cycle, the receiving nexus strobes the C/A information into its receiver latches for decoding. At T0 of the succeeding cycle, the commander asserts data on B(31:00) and, at T3 of the same cycle, the nexus strobes the data into its receiver latches. TAG(2:0) which accompanies the data is coded 101 (write data format). The successsful C/A transfer is confirmed by the receiving nexus with ACK at T0 of the succeeding cycle. The successful data transfer is confirmed by ACK at T0 one cycle later.

Figure 2-41 is a functional timing chart for the Write Masked operation.

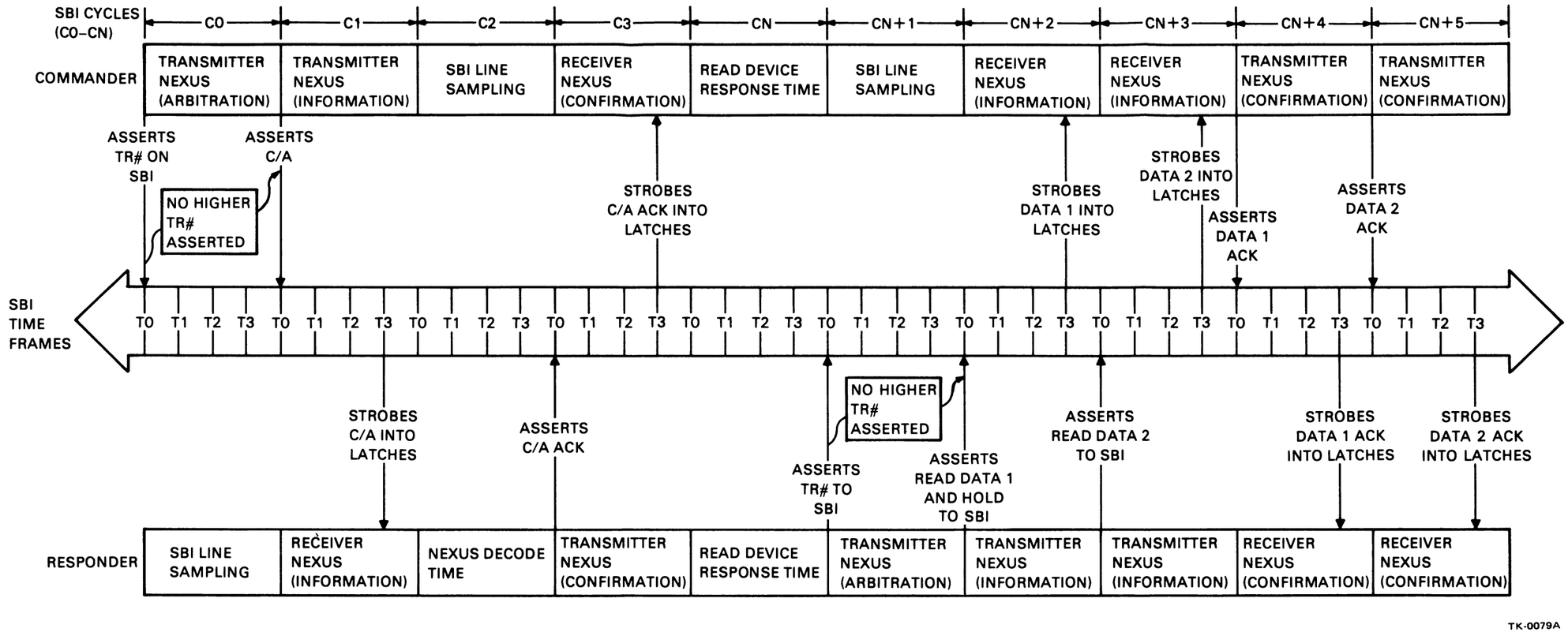


Figure 2-39 Extended Read Timing and Flow
(Sheet 1 of 2)

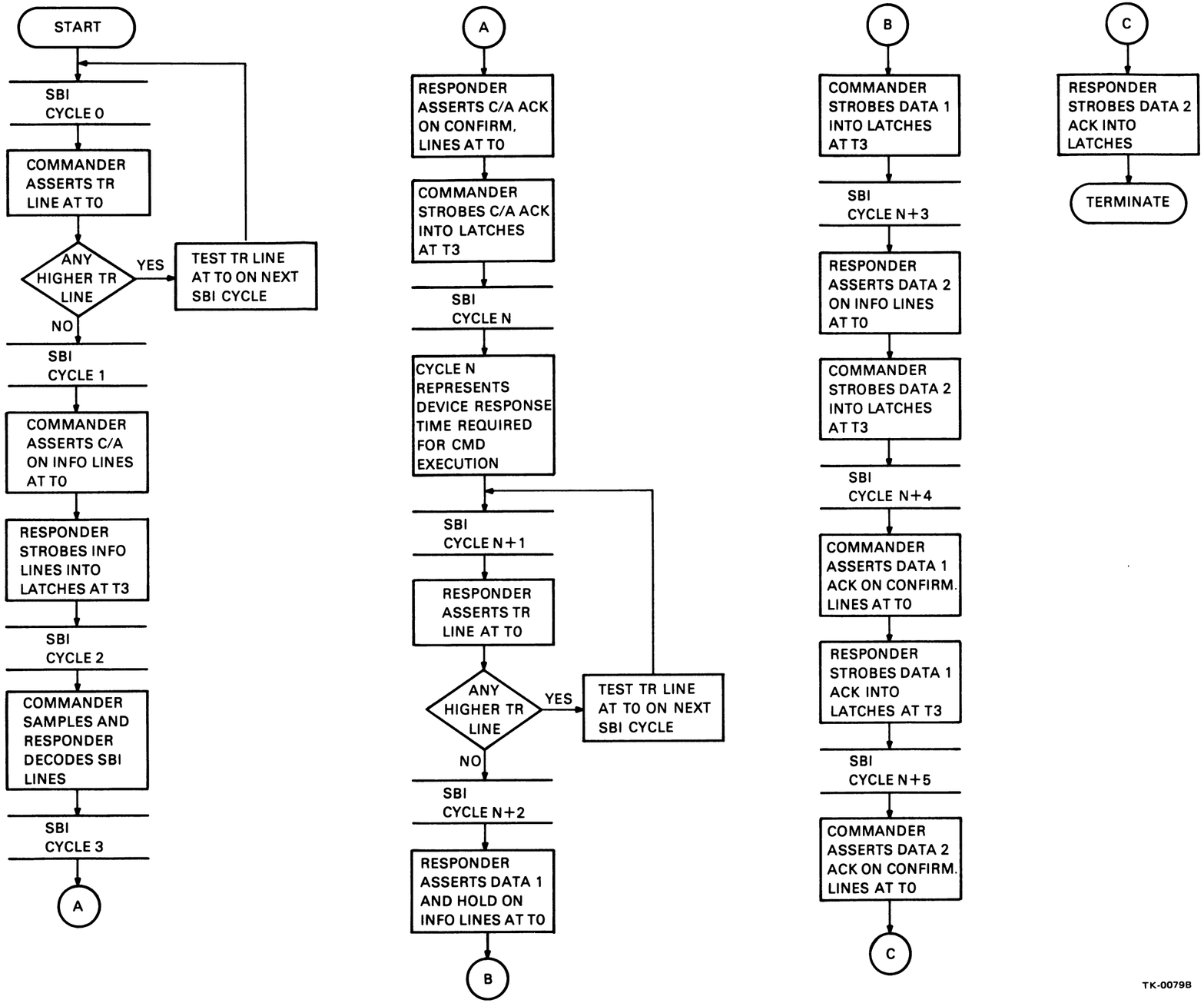


Figure 2-39 Extended Read Timing and Flow (Sheet 2 of 2)

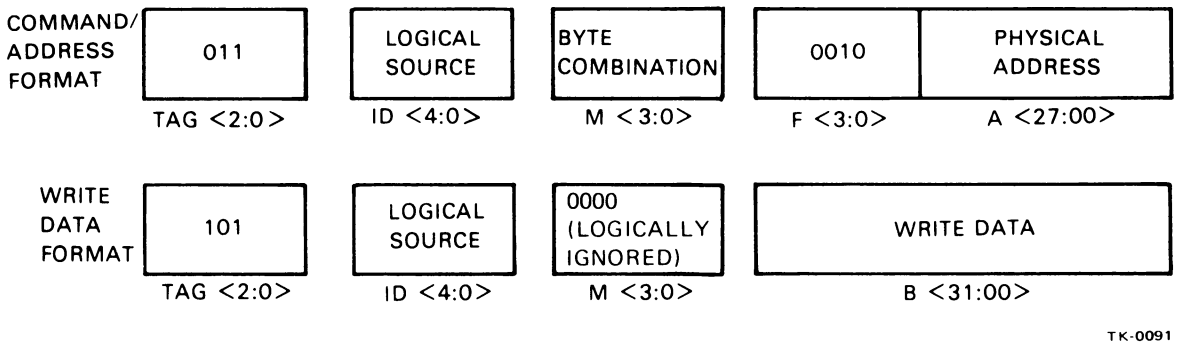
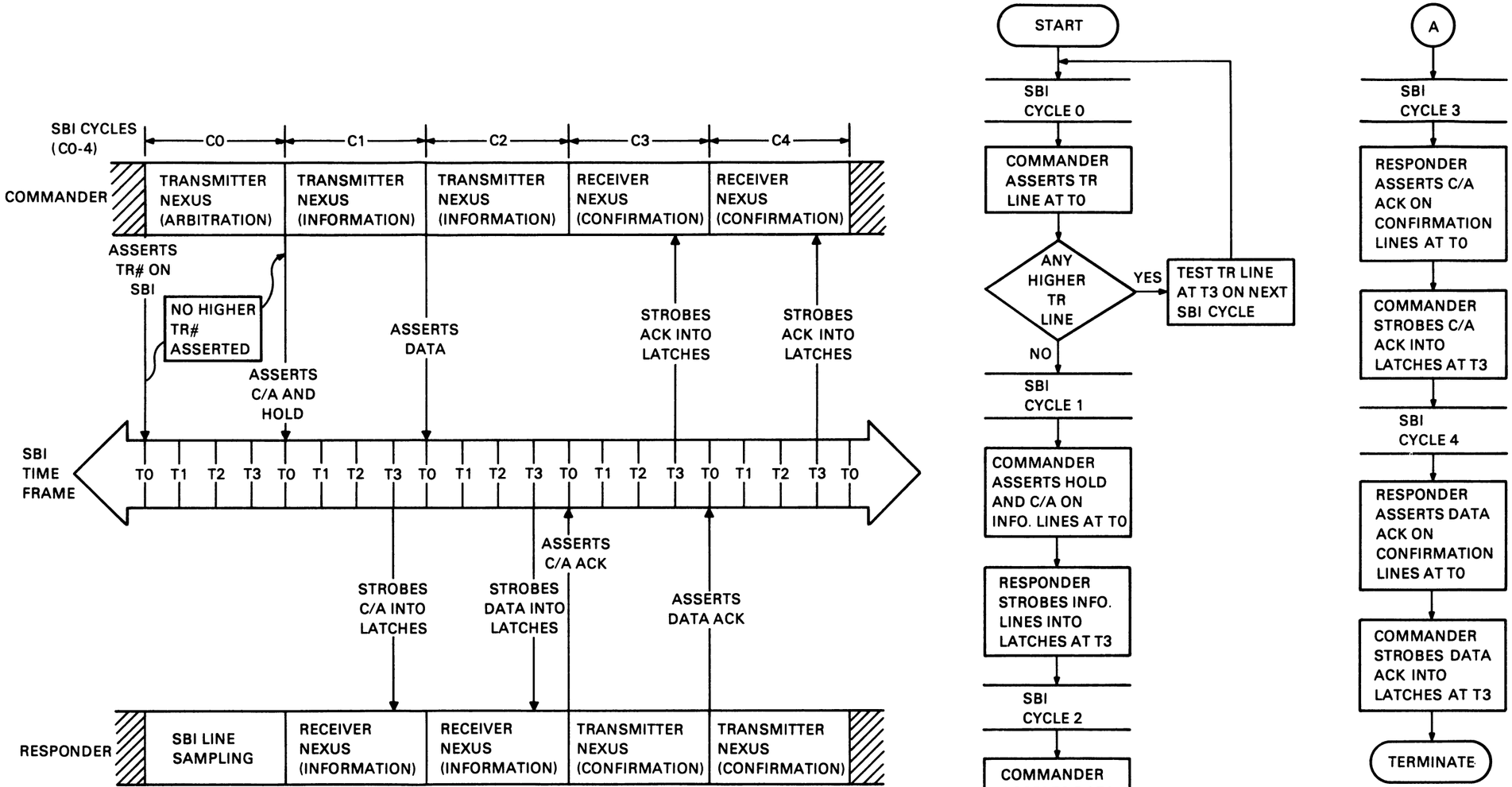


Figure 2-40 Write Masked Function Format



TK-0080

Figure 2-41 Write Masked Timing and Flow

2.3.1.7.4 Extended Write Masked Function – The Extended Write Masked function format is illustrated in Figure 2-42. F(3:0) is coded 1011 to specify the Extended Write Masked function. In the Extended Write Masked transfer, the number of bits written depends on the mask but two SBI data transfer cycles are always required. When the commander gains control of the SBI, it asserts the C/A information at the mask for the first write data at T0. The commander also asserts TR00 to retain control during the succeeding SBI cycle. At T3 of the same cycle, the receiver nexus strobes the C/A information into its latches for decoding. The mask that accompanies the C/A indicates the bytes to be written in the first data longword, corresponding to A00 = 0.

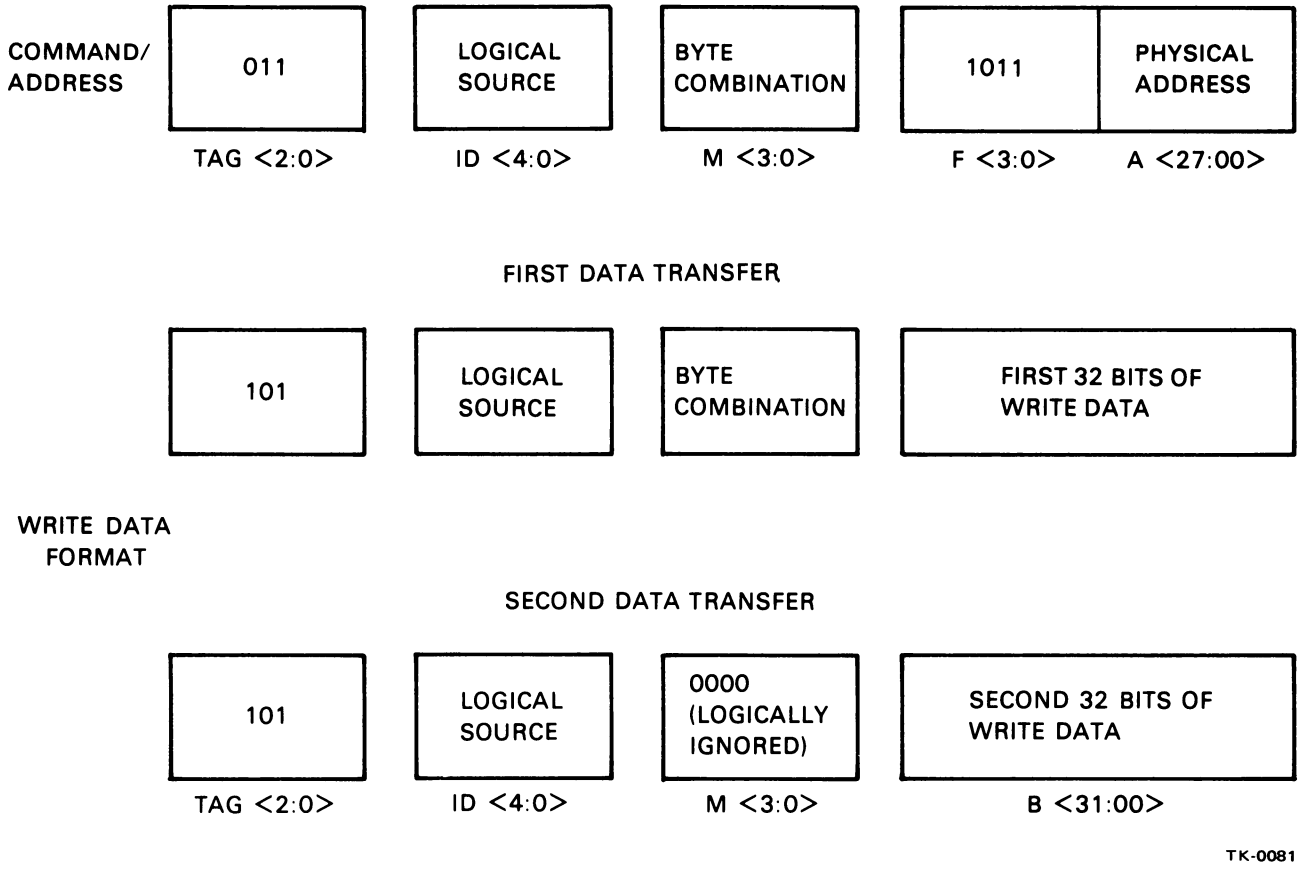


Figure 2-42 Extended Write Masked Function Format

At T0 of the succeeding cycle, the commander asserts data on B(31:00) and codes TAG (2:0) as 101 (write data format). At T3 of the same cycle, the receiver nexus strobes the data into its latches. In addition, the commander holds TR00 asserted to retain SBI Control for the second data word (A00 = 1) transfer. Note that the mask that accompanies the first data word indicates the bytes to be written in the second data word. At the end of this cycle the commander negates TR00.

At T0 of the succeeding cycle, the second data word is asserted on B(31:00), and TAG (2:0) is coded 101. At the same time (T0), the receiver nexus confirms the C/A transfer with ACK, if there is no error. At T3 of the same cycle, the receiver nexus strobes the data into its latches. The mask that

accompanies the second data word is ignored by the receiver nexus; however, the field must be transmitted as 0000. During the two succeeding cycles, the receiver nexus confirms the two data transfers with an ACK in each cycle.

Figure 2-43 is a functional timing chart for the Extended Write Masked operation.

2.3.1.7.5 Interlock Function Description – The interlock function is used to provide coordination between nexus to ensure exclusive access to shared data structures. The Interlock functions operate like the Read and Write functions. However, not all nexus implement the interlock function. Those nexus which do not, respond to the Interlock Read and Write Masked functions exactly as Read and Write Masked functions.

All memory nexus implement the interlock functions and cooperate through the use of the interlock signal (INTLK). The INTLK line is asserted by the commander nexus which issued the Interlock Read Masked function for that SBI cycle following the C/A transfer. The Interlock flip-flop is then set in memory. When the memory nexus confirms the Interlock Read function, it asserts the interlock signal in the same cycle as ACK. With Interlock asserted, the nexus responds with a BSY confirmation to subsequent Interlock Read Masked commands only.

Interlock Read Masked Function Operation – The Interlock Read Masked function format is the same as that shown in Figure 2-36 except that F(3:0) is coded 0100. F(3:0) causes the nexus selected by A(27:00) to retrieve and transfer the addressed data exactly as in the Read Masked operation. In addition, if the selected nexus is memory, its interlock flip-flop is set. With the interlock flip-flop set, memory will assert the SBI interlock line at T0 of the ACK confirmation cycle.

The interlock flip-flop is cleared on receipt of an Interlock Write Masked function. Interlock Read Masked and Interlock Write Masked functions are always paired by commanders. If the flip-flop remains set for more than 102.4 μs, memory assumes that the commander has had a catastrophic error. In this case, memory clears the flip-flop at T0 of the next cycle.

Interlock Write Masked Function Operation – The Interlock Write Masked function format is the same as that illustrated in Figure 2-40 except that F(3:0) is coded 0111, specifying the interlock write function. F(3:0) instructs the nexus selected by A(27:00) to modify the bytes specified by M(3:0) in the addressed storage element, using data transmitted in the succeeding cycle with TAG (2:0) = 101. In addition, if the operation was with memory, the write data clears the interlock flip-flop set by the previous Interlock Read Masked function.

2.3.1.8 Control Group – The control group functions synchronize system activities and provide specialized system communications. The clock functions provide SBI activity synchronization and are described in Paragraph 2.3.1.1. The interlock control, also one of the system communication functions, is described in Paragraph 2.3.1.7. The remaining control lines are described in the following paragraphs.

2.3.1.8.1 DEAD – The DEAD signal indicates a dc power failure in the clock circuits or bus terminating networks. Nexus will not assert any SBI signal while DEAD is asserted. Thus, nexus prevent invalid data from being received while the SBI is in an unstable state.

The assertion of the power supply DC LO to the clock circuits or terminating networks causes the assertion of DEAD. DEAD is asserted asynchronously to the SBI clock and occurs at least 2 μs before the clock becomes inoperative. With power restart, the clock will be operational for at least 2 μs before DC LO is negated. The negation of DC LO negates DEAD.

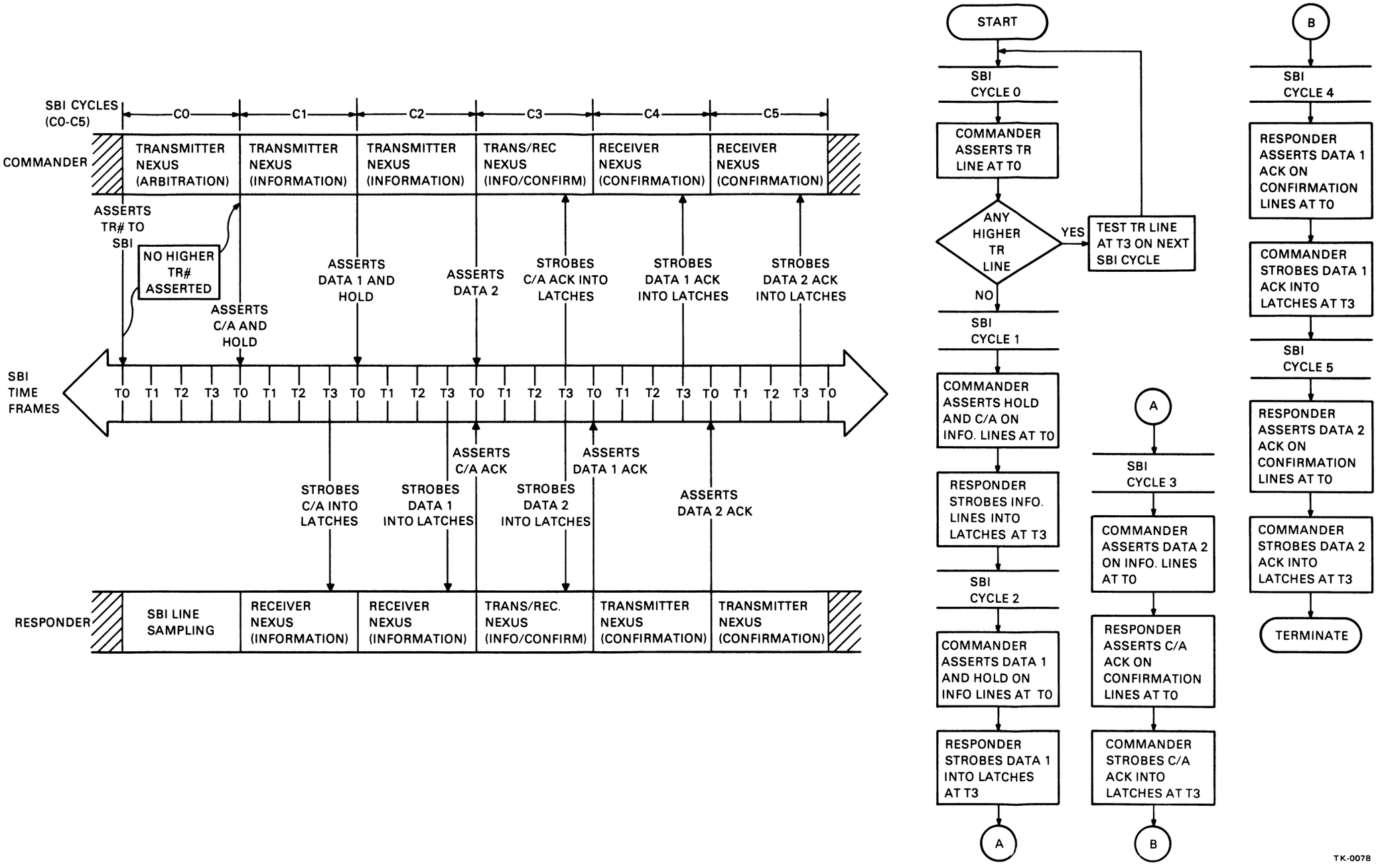


Figure 2-43 Extended Write Masked Timing and Flow

2.3.1.8.2 FAIL – A nexus enables the Fail (FAIL) signal asynchronously to the SBI clock when the power supply AC LO signal is asserted on that nexus. The assertion of FAIL inhibits the CPU from initiating a power-up service routine. FAIL is negated asynchronously with respect to the SBI clock when all nexus that are required for the power-up operation have detected the negation of AC LO. The CPU samples the FAIL line following the power-down routine (assertion of FAIL) to determine if the power-up routine should be initiated. (This case occurs during transient power failures.)

2.3.1.8.3 UNJAM – The UNJAM signal restores (initializes) the system to a known, well defined state. The UNJAM signal is asserted only by the CPU through a console function and is detected by all nexus connected to the SBI. The duration of the UNJAM pulse is 16 SBI cycles and is negated at T0.

For the assertion of UNJAM, the CPU asserts TR00 for 16 SBI cycles. The CPU continues to assert TR00 for the duration of UNJAM and for 16 SBI cycles after the negation of UNJAM. This use of TR00 ensures that the SBI is inactive preceding, during, and after the UNJAM operation. TR00 may be asserted without arbitration.

If asserted, UNJAM is received by every nexus at T3 and a restore sequence is begun. Any current operation of short duration is not aborted, if that operation might leave the nexus in an undefined state. Nexus do not perform operations using the SBI during the assertion of UNJAM. In addition, the nexus is in an idle state (with respect to SBI activity) at the conclusion of the UNJAM pulse.

While UNJAM is asserted, nexus cannot assert FAULT. However, a CPU asserting FAULT prior to UNJAM will continue to do so to preserve the content of the nexus Configuration/Fault Status registers. The restore sequence (UNJAM asserted) should not cause a nexus to pass through any states which will assert any SBI lines. All read commands issued before the UNJAM are cancelled.

In the event of a power failure during UNJAM, some nexus will assert FAIL and/or DEAD. The restore sequence should cause the nexus to clear any existing Alert status bits and subsequently negate ALERT.

2.3.2 SBI Control Logic Description

The logic of the SBI Control is divided on two extended hex boards. Figure 2-44 provides a block diagram of the logic on the M8218 (SBL) module. This board contains logic associated with the lower 16 SBI bits [BUS SBI B(15:00)]. The logic associated with the higher 16 SBI bits [BUS SBI B(31:16)] is contained on the M8219 (SBH) module. A block diagram of this logic is provided in Figure 2-45. Note from these figures, sections of various registers and muxes exist on both boards.

The logic of the SBI Control can also be divided into three basic sections for descriptive purposes. The three sections are: Address Logic, Data Transfer Logic, and ID Bus Logic. The following paragraphs describe each of these sections of logic. The reader should keep in mind that each section is comprised of logic from both the SBL and SBH boards.

2.3.2.1 Address Logic–The following paragraphs describe the address logic of the SBI Control.

PA Register – Figure 2-46 illustrates the address logic of the SBI Control. As seen in this Figure, the PA Address register latches an address from the PA bus at T2 of every CPU cycle. BUFFER FULL H is generated to latch the address and data when an SBI cycle is to be performed. This signal causes the PA register to hold the physical address for the transmit mux or address mux.

Transmit Mux – The transmit mux is shared by the PA register and the Write Data register. For the transmission of a Command/Address, XMIT MUX SEL 1 H remains low to select the PA register for the SBI transceivers. Similarly, XMIT MUX SEL 1 H goes high to send write data and its mask to the SBI transceivers during Write Data transmissions. The transmit mux is disabled when XMIT MUX SEL 0 H is asserted for the transmission of ISR (Interrupt Summary Read) information.

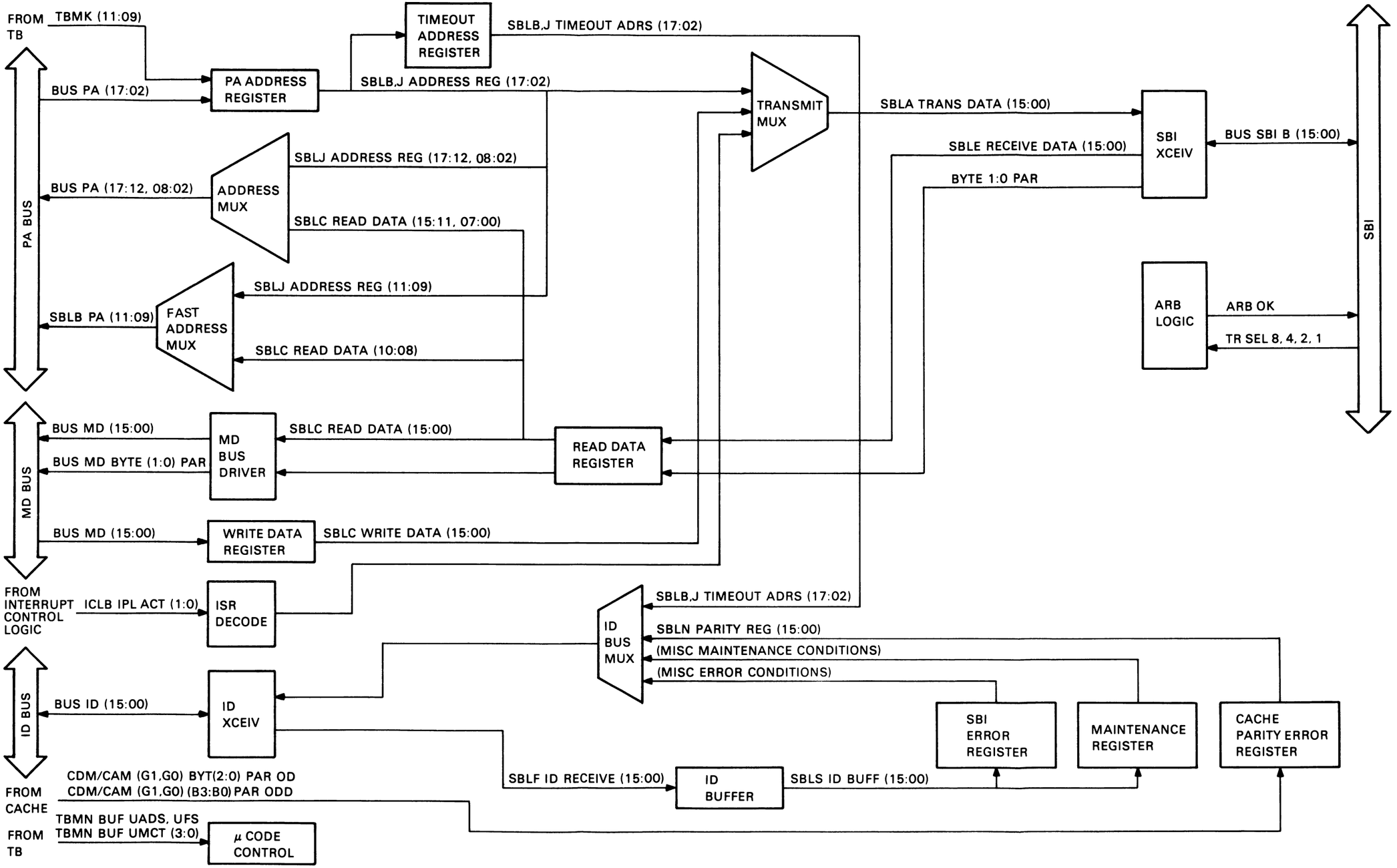


Figure 2-44 SBI Control, Low Bits (SBL)

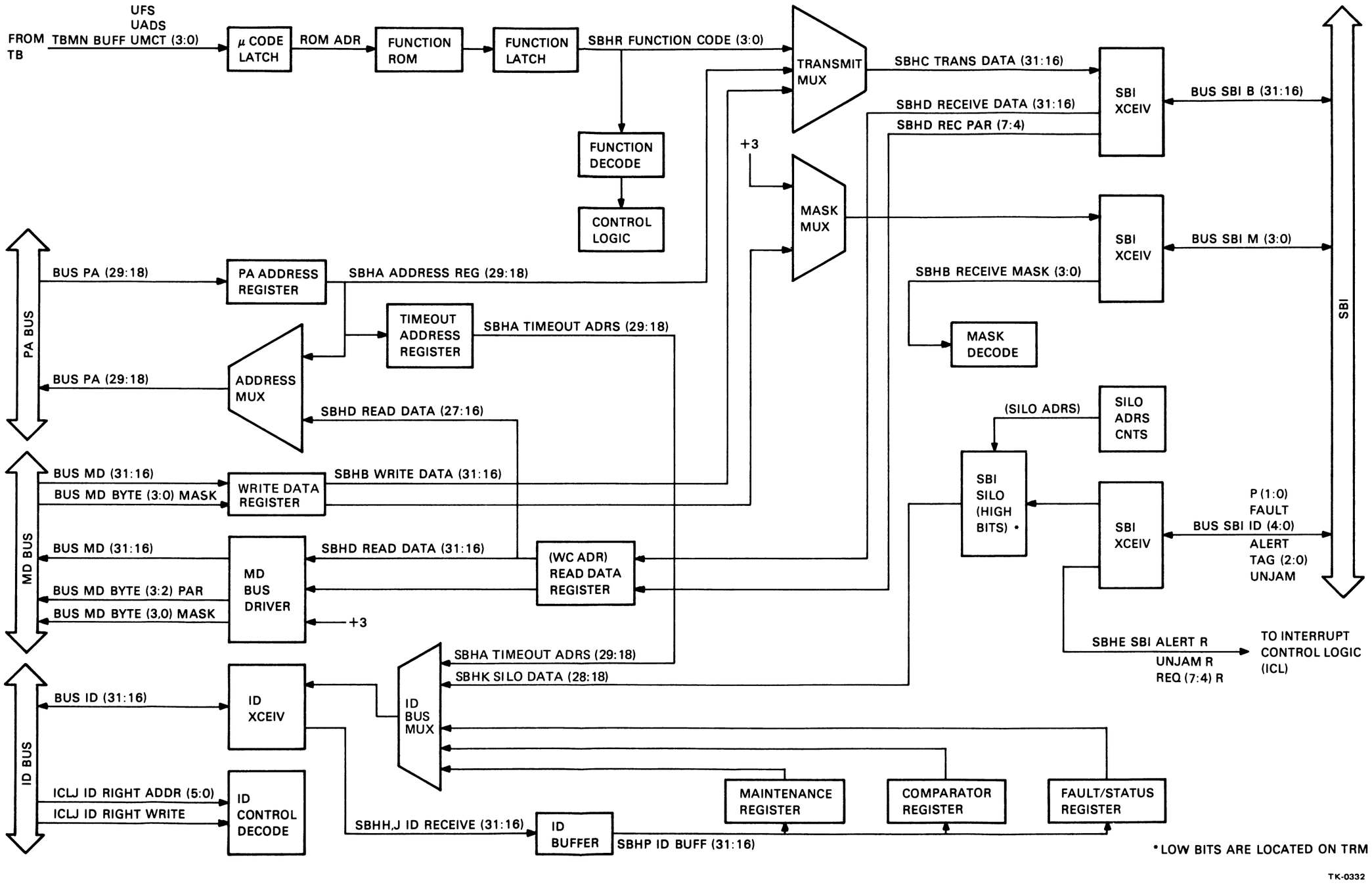


Figure 2-45 SBI Control, High Bits (SBH)

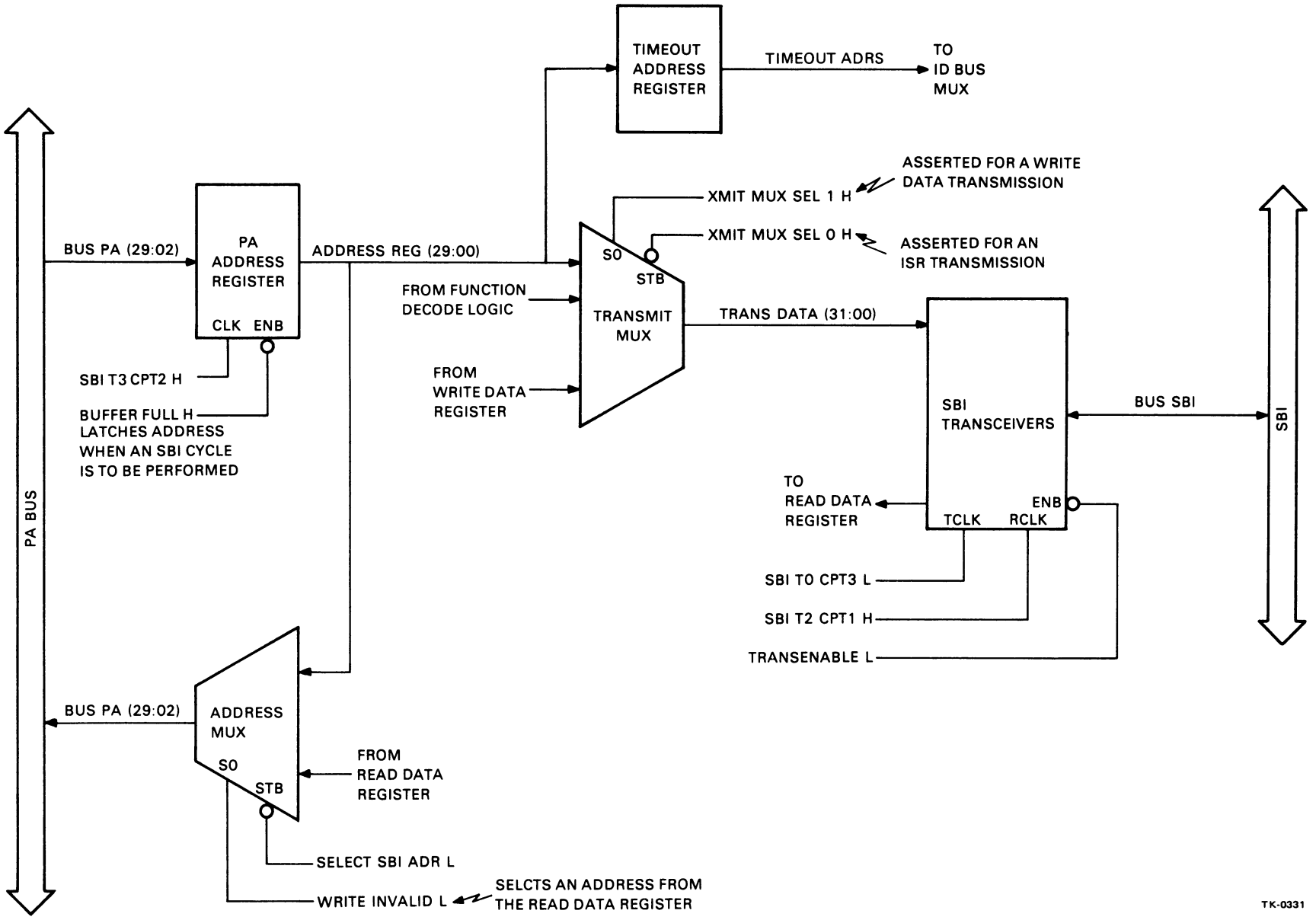


Figure 2-46 Address Logic

Address Mux – The output of the PA register is also connected to the address mux. This mux is shared by the Read Data register. For transmission of any SBI information formats, SELECT SBI ADR L remains unasserted (high) to disable this mux. This creates a high output impedance for the tristate PA bus. In the case of a Cache update, the address from the PA register must be reasserted to the PA bus when the data is retrieved from main memory. For this, SELECT SBI ADR L is asserted and WRITE INVALID L remains negated to select the contents of the PA register for transmission to the PA bus.

An address is selected from the Read Data register when WRITE INVALID L is asserted. WRITE INVALID L is generated by the SBI Control when a nexus (other than the CPU) writes to memory. A cycle to invalidate the Cache entry (if any) is initiated by channeling the SBI address to the PA bus. The actual invalidation cycle only occurs if Cache contains the entry. If the CPU attempts to use the PA bus during the invalidation cycle, the CPU is stalled until the cycle is complete.

SBI Transceivers – The SBI transceivers in Figure 2-46 are used to transmit Command/Address, Write Data, and Interrupt Summary Read formats to the SBI. They are also used to latch Read Data and Interrupt Summary Response formats from the SBI. All information latched from the SBI is channeled to the Read Data register whose output is connected to MD bus drivers and the address mux.

Information is latched from the SBI and presented to the Read Data register at T3 every SBI cycle via the SBI transceivers. Information is transmitted at SBI T0 only if TRANSENABLE L is generated. TRANSENABLE L is generated as a result of SBI function decode (Paragraph 2.3.3) for the transmission of a command/address, write data, or a maintenance format.

Timeout Address Register – The address logic includes a Timeout Address register which latches the address from the PA register when a timeout occurs on the SBI as a result of a Command/Address transmission by the SBI Control. The output of the Timeout register is conrp nected to the ID bus mux which makes it readable over the ID bus. The register format is described in Paragraph 2.3.2.6. Paragraph 2.3.2.3 provides a description of the ID mux logic.

2.3.2.2 Data Transfer Logic – The following paragraphs describe the data transfer logic of the SBI Control.

Write Data Register – Figure 2-47 illustrates the data transfer logic of the SBI Control. As seen in this Figure, the Write Data register latches data and its mask from the MD bus at T3 of every SBI cycle. Just as for the PA Register, BUFFER FULL H is generated to latch the data when an SBI cycle is to be performed. This signal causes the Write Data register to hold the data and mask for the transmit mux and mask mux.

Transmit Mux (See paragraph 2.3.2.1).

Mask Mux – The mask mux selects the mask field for an SBI transmission. MASK MUX SEL H is generated to select a mask from the Write Data register. (This mask is originally generated by the CPU data path and placed on the MD bus.) For the transmission, which requires a mask of 0000, DISABLE MASK MUX H is asserted to disable the mux. Likewise, a mask of 1111 can be selected by the negation of DISABLE MASK MUX H and MASK MUX SEL H. Table 2-7 lists the required mask for various transmissions.

Read Data Register – At T3 of every SBI cycle, an address or data is latched from the SBI and enabled to the Read Data register. One parity bit is generated for every four bits latched and likewise input to the register. The Read Data register is loaded with the latched address or data at the following T1 of every SBI cycle. The address or data is then available to the MD bus drivers for transfer to the MD bus or the address mux for transfer to the PA bus.

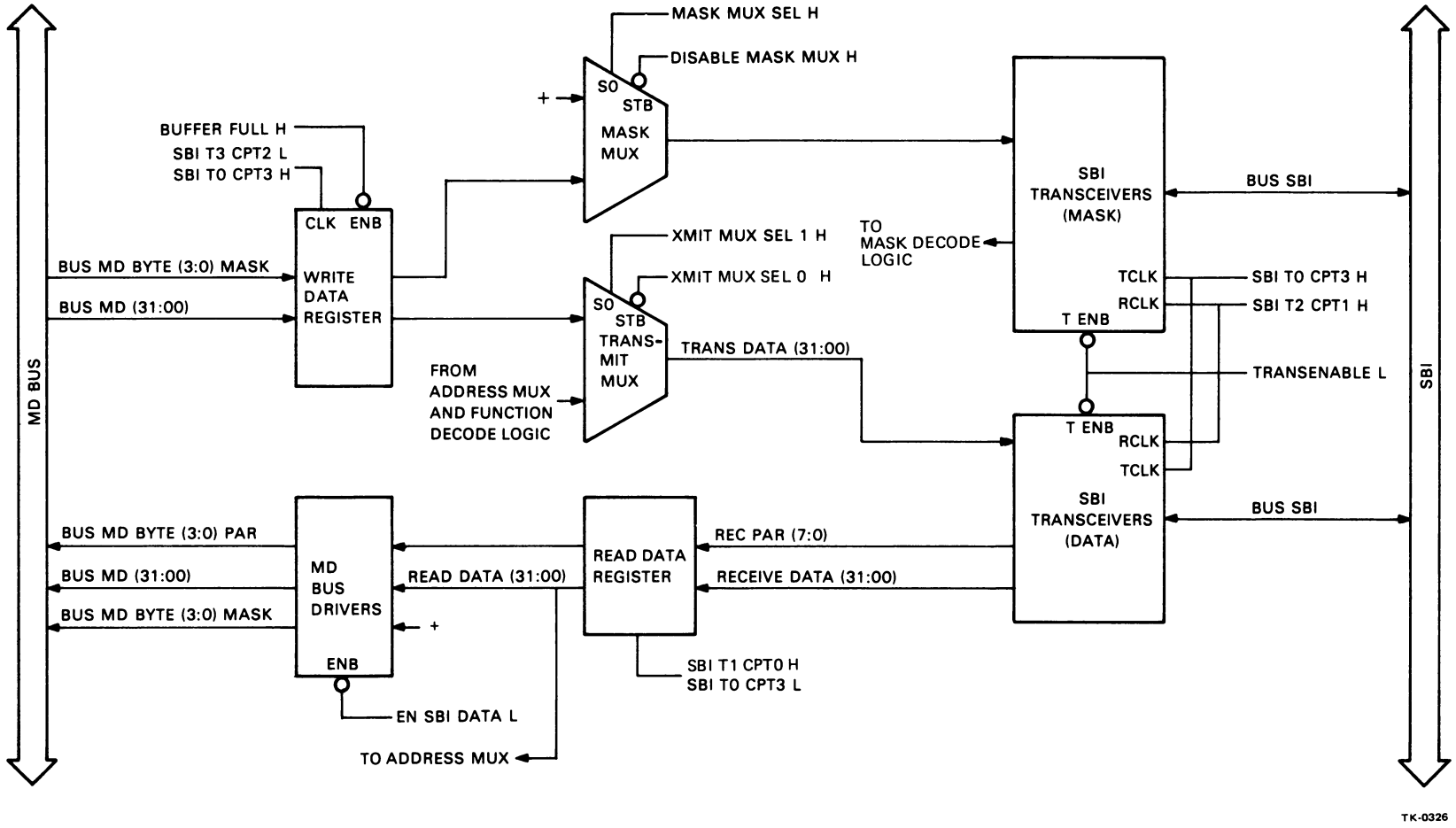


Figure 2-47 Data Transfer Logic

Table 2-7 Mask Mux Selection

Operation	Format	Selected Mask
Read Masked	Command/Address	From Write Data register
Extended Read	Command/Address	0000
Write Masked	Command/Address Write Data	From Write Data register 0000
Extended Write Masked	Command/Address Write Data (first) Write Data (second)	From Write Data register From Write Data register 0000
Extended Write Masked (full)	Command/Address Write Data (first) Write Data (second)	1111 1111 0000
IB References	- Interrupt Summary Read	1111 0000

MD Bus Drivers – The MD bus drivers transmit data from the Read Data register to the MD bus. The signal EN SBI DATA L is generated to enable these drivers. When EN SBI DATA L is negated, the MD bus drivers are inhibited. This frees the MD bus for data transfers between the CPU data path (or instruction buffer) and the Write Data register.

2.3.2.3 ID Bus Logic – Figure 2-48 illustrates the ID bus logic of the SBI Control. The following paragraphs describe each section of this logic. A detailed description of each ID register of the SBI Control is also included.

ID Transceivers – The ID bus transceivers enable ID bus data to the input of the ID buffer as long as EN ID DRIVERS L remains unasserted (high). With the generation of EN ID DRIVERS L, the transceivers become transmitters for the ID register data selected by the ID bus mux. The generation of this signal is controlled by the ID control decode logic and the ID RIGHT WRITE signal.

ID Buffer – As seen in Figure 2-48, the ID buffer latches data received by the ID transceivers for input to the ID registers. Control signals from the ID control decode logic are generated to enable the proper ID registers to receive the data. ID data is latched by the ID buffer at CPT1 (high bits) and CPT2 (low bits).

ID Bus Mux – The ID bus mux is used to select an ID register for transmission to the ID bus via the ID transceivers. The select lines, ID ADDR (2:0), are generated in the ID control decode logic. The ID bus mux is always enabled.

ID Control Decode – The ID control decode logic receives and decodes an ID bus address and control signal during every CPU cycle. The decode generates control signals for the ID transceivers, buffer, and bus mux. Table 2-8 summarizes the resultant data flow for each condition of the control line. Table 2-9 lists the addressable ID registers of the SBI Control.

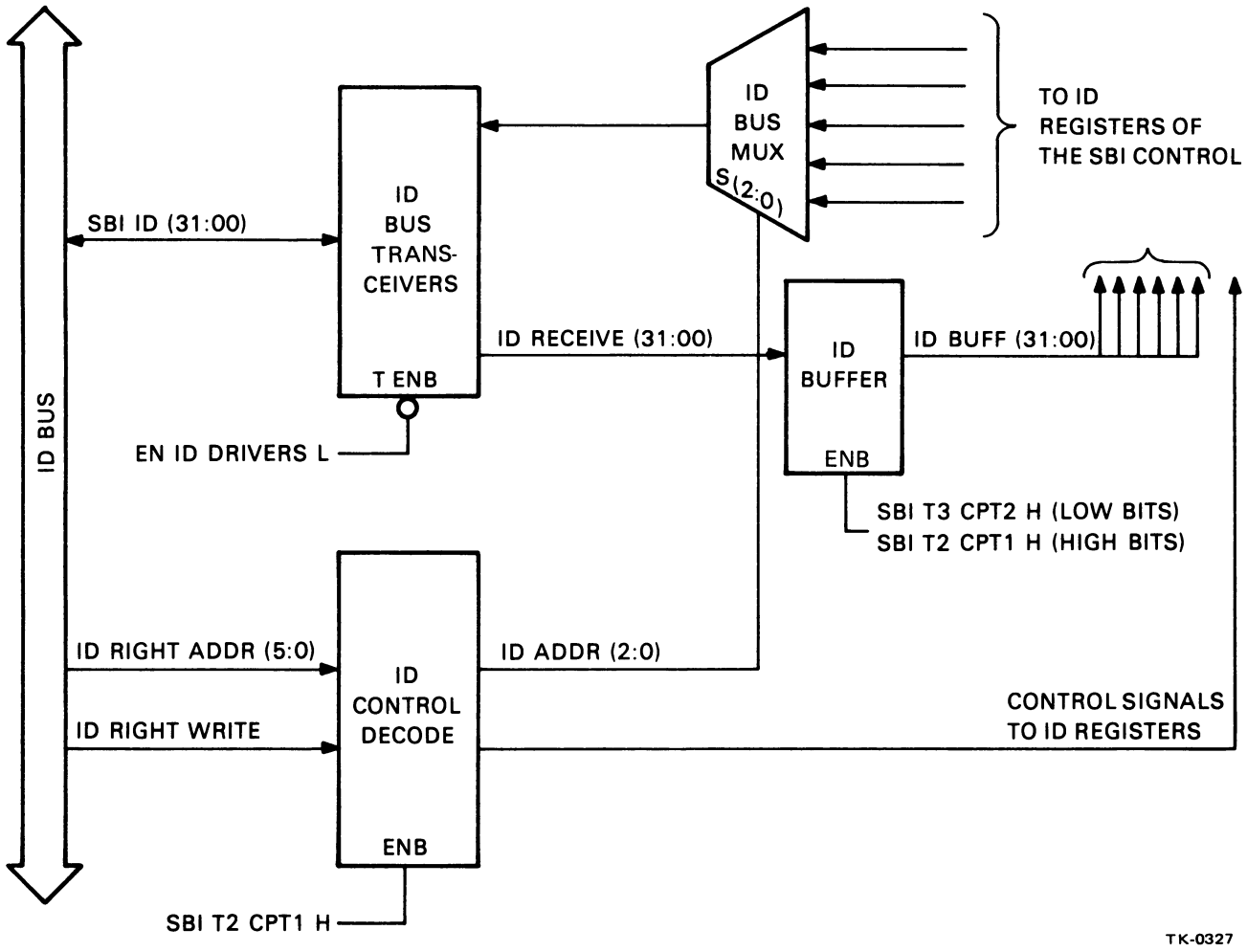


Figure 2-48 ID Bus Logic

Table 2-8 ID Data Flow

ID RIGHT WRITE L	Data Flow
1	ID Bus ← Addressed ID register
0	Addressed ID Register → ID Bus

Table 2-9 ID Register Addresses (of the SBI Control)

ID Right Addr		ID Register
Binary **543210**	**Hex**	
010010	18	SBI Silo
010011	19	SBI Error
010100	1A	Timeout Address
010101	1B	Fault/Status
010110	1C	SBI Silo Comparator
010111	1D	Maintenance
011000	1E	Cache Parity

2.3.2.3.1 SBI Silo – The SBI silo is a read-only register file which provides temporary storage of various SBI signals for the last 16 SBI cycles. Figure 2-49 illustrates the SBI signals stored. This information is latched during every SBI cycle and held for 16 cycles. Thus the silo always contains records of the previous 16 SBI cycles.

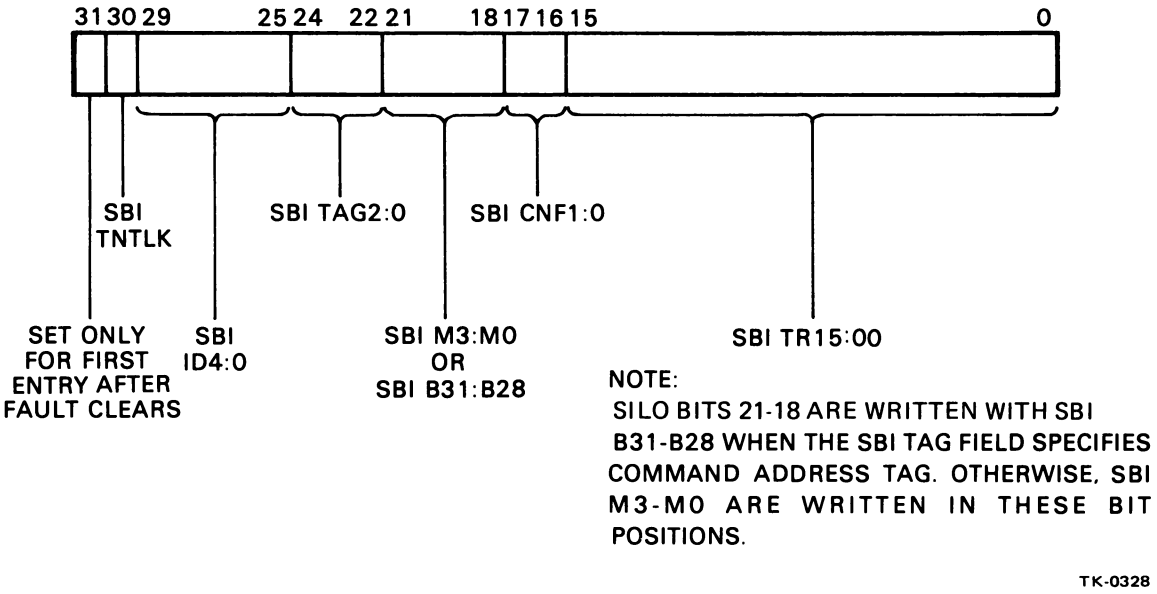


Figure 2-49 SBI Silo Data

Without the assertion of FAULT, the SBI information is written in the silo and its 4-bit address counter is advanced. The assertion of FAULT by any nexus prevents writes in the silo and inhibits advancement of its address counter. The silo remains locked until FAULT is cleared. When FAULT is deasserted, bit 31 is set for the first silo write. This provides a fault marker for diagnostics.

Writing in the silo may also be inhibited through the use of the SBI Comparator register. The SBI Comparator register is described in Paragraph 2.3.2.3.2.

Figure 2-50 illustrates the SBI FAULT and silo timing. As shown in this Figure, when a parity error occurs, SBHL SBI FAULT H is asserted in the following cycle. The generation of this signal sets FAULT on the SBI (BUS SBI FAULT L). With FAULT asserted on the SBI, SBHE SBI FAULT R H is generated at T3 of the same SBI cycle to inhibit writing in the silo and incrementing the silo address counter. This sequence ensures the assertion of FAULT to the SBI before the silo is locked. SBHL SEND FAULT 1 H is then generated so that the CPU continues to assert FAULT which inhibits the silo and latches all Fault Status registers.

The ID bus is used to read the silo and clear the Fault register. With the silo address counter locked, a silo read operation is initiated and the contents of the locked location are transferred to the ID bus. This reasserts SBHK SILO COUNT EN H which frees the silo address counter. With SBHK SILO COUNT EN H again enabled, the address counter is incremented at the next counter clock when the silo is read. This provides the contents of the next silo location for an ID bus transfer. Each subsequent read increments the counter providing the contents of the next silo location.

When sufficient information has been read, SBHK SILO COUNT EN H again goes low to inhibit the address counter. Once the Fault register is cleared, SBHL SEND FAULT 1 H is dropped to deassert SBHL SBI FAULT H. Without SBHL SBI FAULT H, FAULT is removed from the SBI at the following T0 (BUS SBI FAULT L goes high). SBHE SBI FAULT R H also goes low at the next T3. Coincidentally, the address counter is again enabled for writes to the silo, provided no other nexus is asserting FAULT. Note the writes begin at the silo address which follows the last address read.

2.3.2.3.2 SBI Comparator Register – The SBI Comparator register is a maintenance tool which provides another means to lock the SBI silo other than the assertion of FAULT on the SBI. The Comparator register may lock the silo under two modes of operation.

The first mode of operation is the unconditional lock mode. In this mode the SBI silo can be locked anytime within 15 cycles after writing in the Comparator register. The number of cycles is dictated by contents of the count field. This field is always set in the 1's complement form of the desired number of cycles and is incremented automatically for each SBI cycle. The silo locks when the count field is equal to all 1s. For example, to unconditionally lock the silo in one cycle, the count field is loaded with 1110.

The second mode of operation is the conditional lock mode. In this mode, the incrementation of the count field is started only after certain conditions on the SBI are detected. Once these conditions are detected, the count field is incremented and used just as it is during unconditional lock mode operation (described above). Table 2-10 lists the selectable lock condition(s). The Maintenance ID bits referenced in the table are located in the Maintenance register (Paragraph 2.3.2.3.7). Note also that for compare mode ID. TAG. Cmd Fnc, the command/mask field is compared against SBI B (31:28) if the compare tag field indicates Command/Address (equals 011). In this case the field is interpreted as a command function. Otherwise the command/mask field is compared against SBI M(3:0) and the field is assumed to contain a mask.

In either mode of operation, the SBI silo is unlocked by loading the count field of the Comparator register with a number other than 1111.

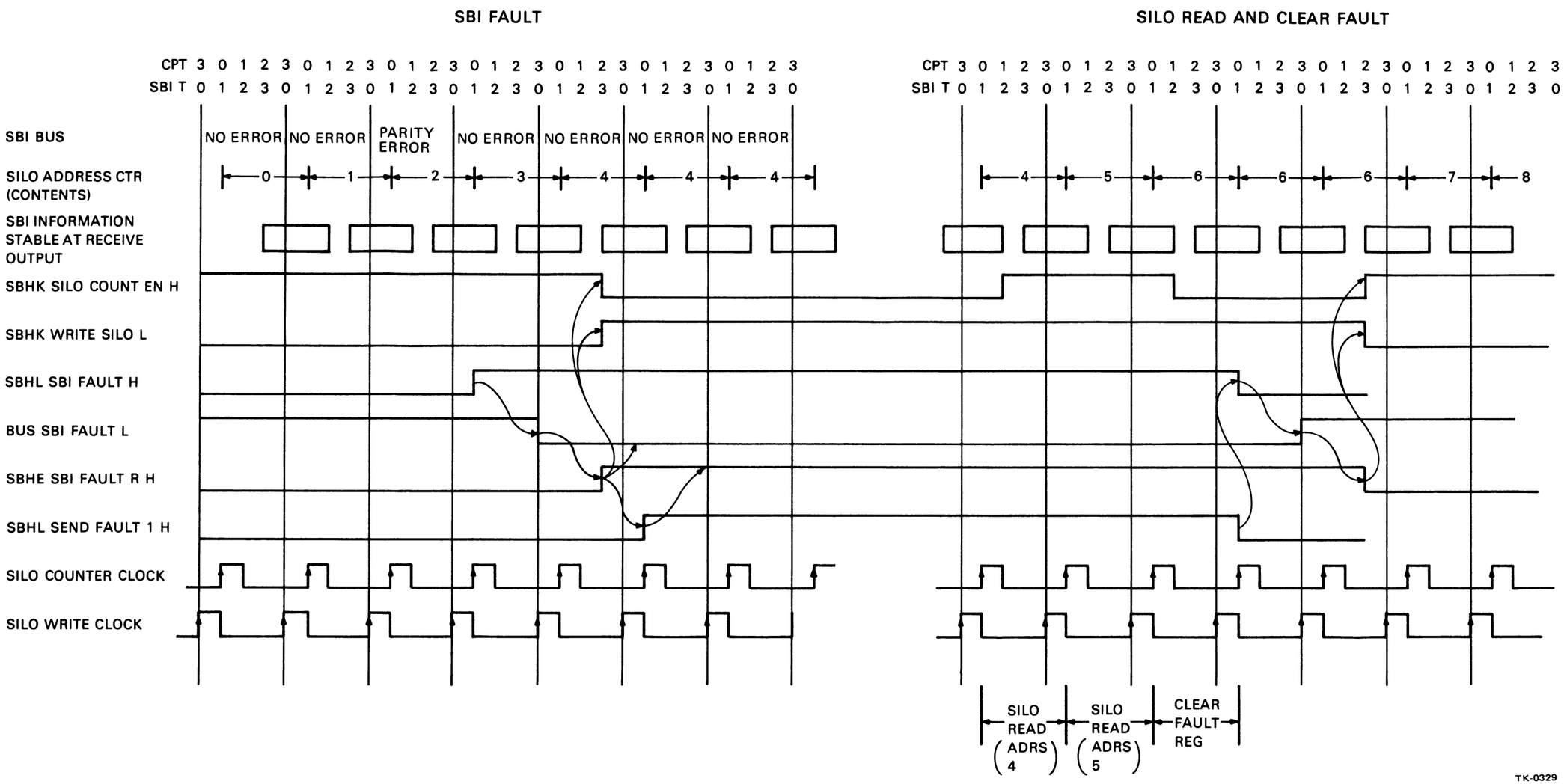


Figure 2-50 SBI Fault and Silo Timing

Table 2-10 Conditional Lock Codes

Bits 28	Bits 27	Compare Mode	Lock Conditions
0	0	No Compare	-
0	1	ID only	SBI ID = Maintenance ID
1	0	ID.TAG	SBI ID = Maintenance ID and SBI TAG = Comparator TAG
1	1	ID.TAG.Cmd Fnc	SBI ID = Maintenance ID and SBI TAG = Comparator TAG and either SBI Function [SBI B(31:28)] = Comparator Command or SBI Mask [SBI M(3:0)] = Comparator Mask.

The Silo Comparator register is located on the SBL board. Figure 2-51 illustrates the register format. The mode of operation is selected by setting or clearing bit 29. This bit is set for unconditional lock and cleared for conditional lock. Bits 28 and 27 contain the conditional lock codes. Bits (22:20) contain the compare tag and bits (26:23) contain the compare command or compare mask. The count field is contained in bits (19:16). If COMP silo lock (bit 31) and silo lock interrupt enable (bit 30) are set, an interrupt request is generated when the count field equals 1111.

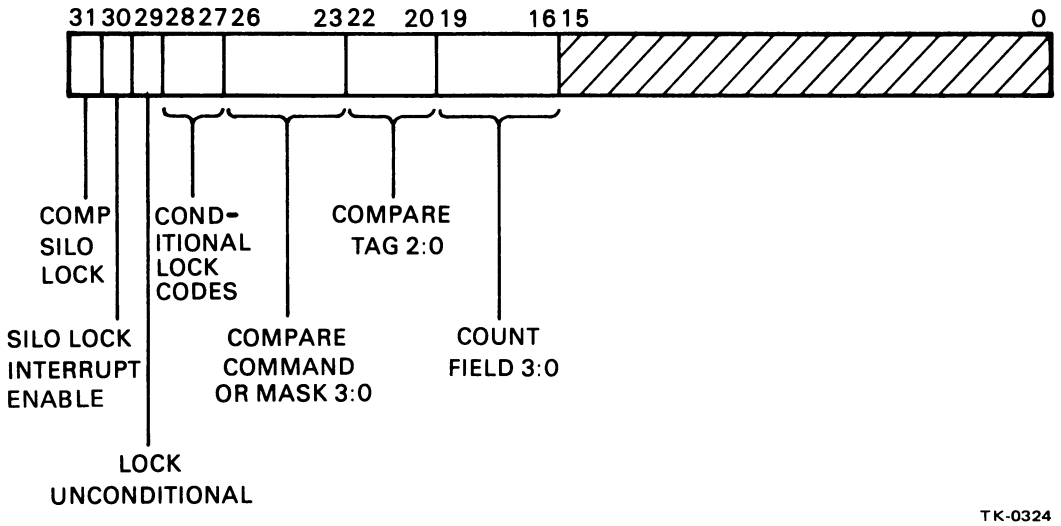


Figure 2-51 Silo Comparator Register

2.3.2.3.3 Timeout Address Register – The Timeout Address register is a read-only holding register which latches the transmitted physical address when BUFFER FULL L is asserted or a timeout occurs on the SBI. The address remains latched until the timeout error bit (bit 12) in the SBI Error register is cleared (Paragraph 2.3.2.3.5). The address is not latched, however, if the timeout occurs during a data fetch for the instruction buffer. This register is addressable over the ID bus.

Figure 2-52 illustrates the Timeout Address register format. Bits (27:00) contain the physical address of the timeout. Bit 29 is set if the reference underwent a hardware protection check and remains unasserted if the reference was not subject to a hardware protection check. Bits 31 and 30 provide the mode of the reference that resulted in the timeout.

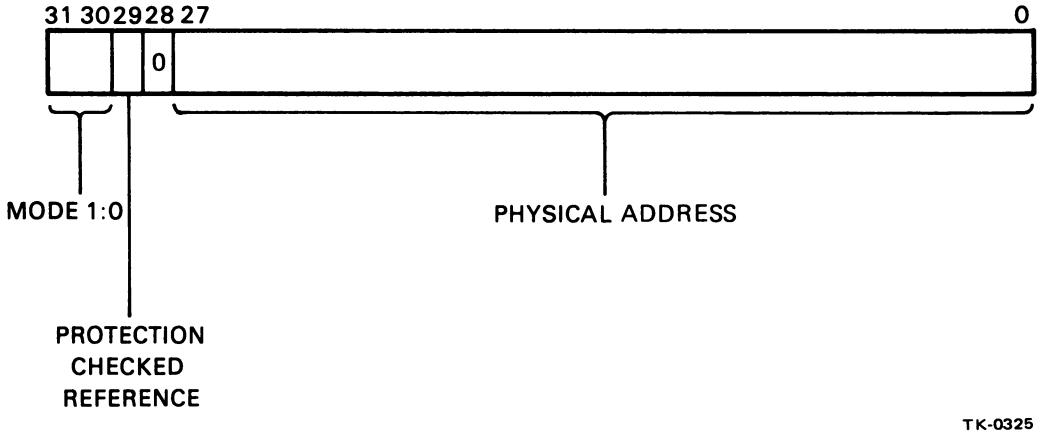


Figure 2-52 Timeout Address Register

2.3.2.3.4 Cache Parity Error Register – Although located on the SBL board, the Cache Parity Error register stores results from parity checks on the CAM and CDM boards. Figure 2-53 illustrates the register format. Each bit is described in Table 2-11.

2.3.2.3.5 SBI Error Register – The SBI Error register provides a record of various SBI error conditions. This register is located on the SBL board. Figure 2-54 illustrates the register format. Each bit is described in Table 2-12.

2.3.2.3.6 Fault/Status Register – The Fault/Status register provides bits to indicate conditions which cause the assertion of FAULT. Figure 2-55 illustrates the register format. Each bit is described in Table 2-13.

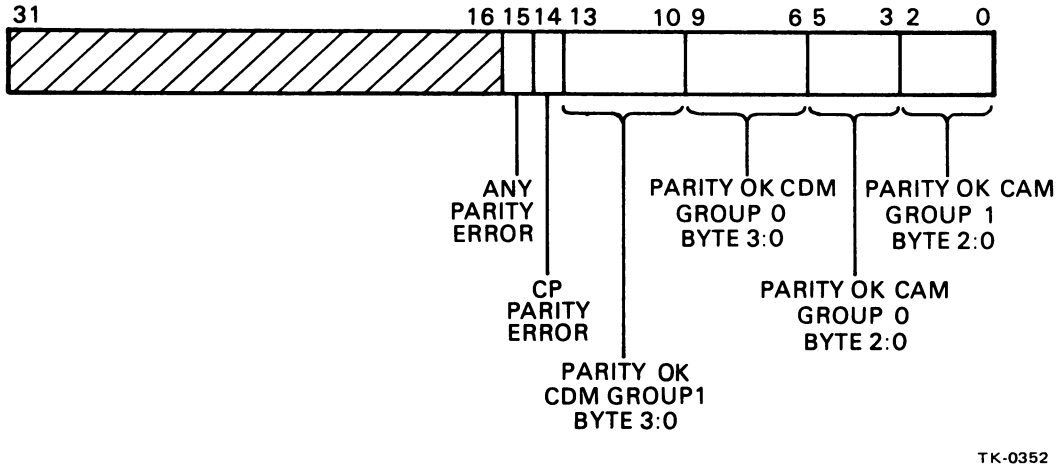


Figure 2-53 Cache Parity Error Register

Table 2-11 Cache Parity Error Register

Bit	Function	Description
15	Any Parity Error	If set, indicates a Cache parity error has been detected on an IB or CP read operation. This bit is read/write 1 to clear. When this bit is cleared, bit (14:00) is also cleared.
14	CP Parity Error	With bit 15 set, this bit indicates whether the Cache parity error occurred on a reference by the CP or IB. 1 = CP 0 = IB
13:00	Parity OK	With bit 15 set, these bits identify the Cache bytes which do not contain a parity error (1 = no error, 0 = error). These bits are cleared when bit 15 is cleared.

NOTE

If this register contains a parity error for the instruction buffer, this register is automatically cleared when the instruction buffer is flushed.

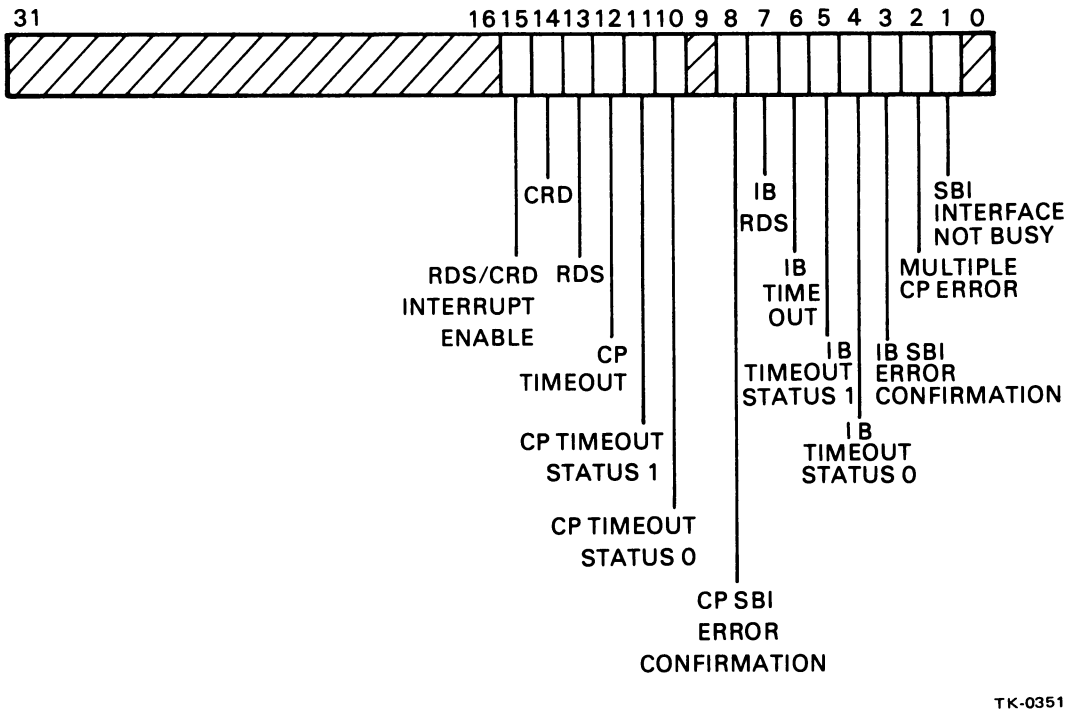


Figure 2-54 SBI Error Register

Table 2-12 SBI Error Register

Bit	Function	Description
15	RDS/CRD Interrupt Enable	If set and an RDS or CRD occurs, an interrupt request is initiated. This bit is read/write.
14	CRD	Set when CRD (Corrected Read Data) is returned to the CPU. An interrupt request is initiated if bit 15 is also set. This bit is read/write 1 to clear.
13	RDS	Sets when RDS (Read Data Substitute) is returned to the CPU. An interrupt request is initiated if bit 15 is also set. This bit is read/write 1 to clear.
12	CP Timeout	Sets when a timeout occurs for a CPU requested cycle. While this bit is set, an interrupt is requested. This bit is read/write 1 to clear. When cleared, bits 11, 10, and 8 are also cleared.

Table 2-12 SBI Error Register (Cont)

Bit	Function	Description
11, 10	CP Timeout Status	These bits describe the type of timeout and are valid only if bit 12 is set. The types of timeouts are listed below. These bits are read-only. **Bits 11 10** | **Type of Timeout** 0 0 | Device No Response 0 1 | Device Was Busy 1 0 | Waiting for Read Data 1 1 | (Not used)
8	CP SBI Error Confirmation	Sets when a CP requested cycle receives an error confirmation on a Command/Address transmission. While this bit is set, an interrupt is requested. This bit is read-only.
7	IB RDS	Sets if an RDS is received while the SBI Control is fetching data for the instruction buffer. This bit is read/write 1 to clear. It is also cleared when the instruction buffer is flushed (cleared).
6	IB Timeout	Sets when a timeout occurs during a cycle requested by the instruction buffer. While this bit is set, an interrupt is requested. This bit is read/write 1 to clear. It is also cleared when the instruction buffer is flushed (cleared). When cleared, bits 5 and 4 are also cleared.
5, 4	IB Timeout Status	These bits describe the type of timeout and are valid only if bit 6 is set. The types of timeouts are listed below. These bits are read-only. **Bits 5 4** | **Type of Timeout** 0 0 | Device No Response 0 1 | Device Was Busy 1 0 | Waiting for Read Data 1 1 | (Not used)

Table 2-12 SBI Error Register (Cont)

Bit	Function	Description
3	IB SBI Error Confirmation	Sets when a cycle requested by the instruction buffer receives an error confirmation on a Command/Address transmission. While this bit is set, an interrupt is requested. This bit is read-only.
2	Multiple CP Error	Sets when a CP timeout or CP SBI error confirmation occurs and the CP timeout or CP SBI error confirmation bit is already set. This bit is also cleared when bit 12 is cleared.
1	SBI Interface Not Busy	This bit is set when the SBI control is not busy executing an SBI transfer.

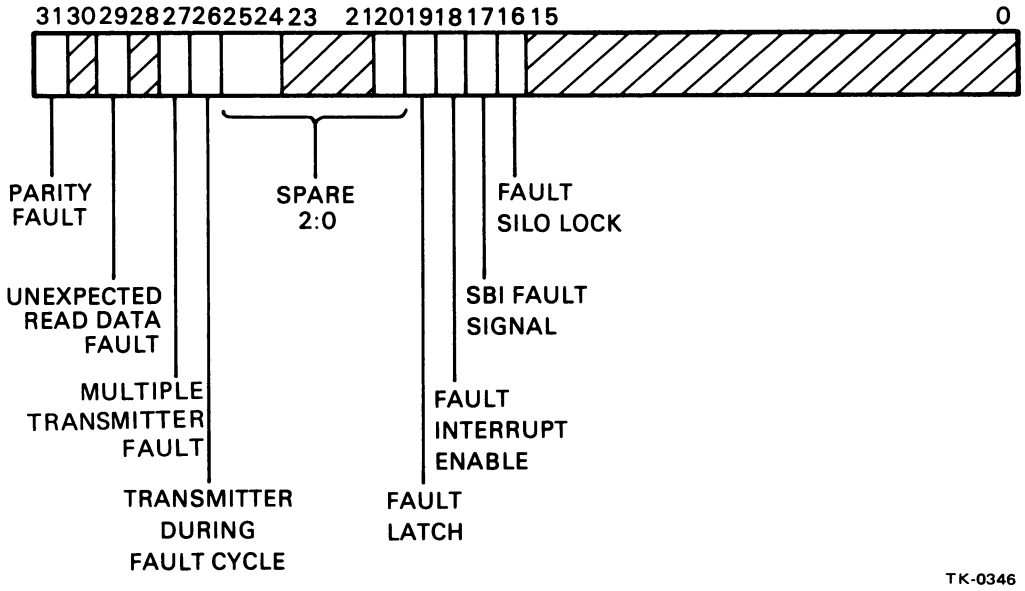


Figure 2-55 FAULT/Status Register

Table 2-13 Fault/Status Register

Bit	Function	Description
31	Parity Fault	If set, indicates an information path parity error was detected on the SBI. This bit is read-only.
29	Unexpected Read Data Fault	If set, indicates a read-type response has been received without a previously issued read masked, extended read masked, or interlock read masked command. This bit is read-only.
27	Multiple Transmitter Fault	If set, indicates the CPU has detected multiple transmitters in the same cycle. This bit is read-only.
26	Transmitter During Fault Cycle	If set, indicates the SBI Control was the transmitter during the cycle in which FAULT was asserted. This bit is read-only.
25, 24, 20	Spare Bits	Unused by hardware.
19	Fault Latch	If set, indicates the SBI Fault signal has been asserted. While this bit is set, the CPU asserts FAULT on the SBI. An interrupt request is initiated when this bit is set if bit 18 is also asserted. This bit is read/write 1 to clear.
18	Fault Interrupt Enable	If set, enables an interrupt when bit 19 is asserted. This bit is read/write.
17	SBI Fault	If set, indicates SBI FAULT is being asserted on the SBI. This bit is read-only.
16	Fault Silo Lock	If set, indicates the SBI silo has locked due to the assertion of SBI FAULT. (If the Comparator register locked the silo simultaneously, a bit in the Comparator register is also set (Paragraph 2.3.2.5). This bit is read/write 1 to clear.

2.3.2.3.7 Maintenance Register - The Maintenance register contains maintenance and status information of Cache and the SBI Control for diagnostic use. Half of the maintenance register is located on the SBL board and half is located on the SBH board. Figure 2-56 illustrates the register format. Each bit is described in Table 2-14.

2.3.2.4 SBI Cycle Initiation Logic - An SBI cycle is initiated by the SBI Control for any of the following, provided SBI cycles are not prohibited (conditions which prohibit SBI cycles are listed below):

1. A write by the CP.
2. A read by the CP in which a Cache miss occurs. The SBI cycle is inhibited in this case if a Cache parity error occurs.
3. An Interrupt Summary Read command (ISR).
4. An instruction buffer request during an ALLOW.IB cycle in which a Cache miss occurs. The SBI cycle is inhibited in this case if a Cache parity error occurs.
5. An Interlock Read command. The SBI cycle is inhibited in this case if a Cache parity error occurs.

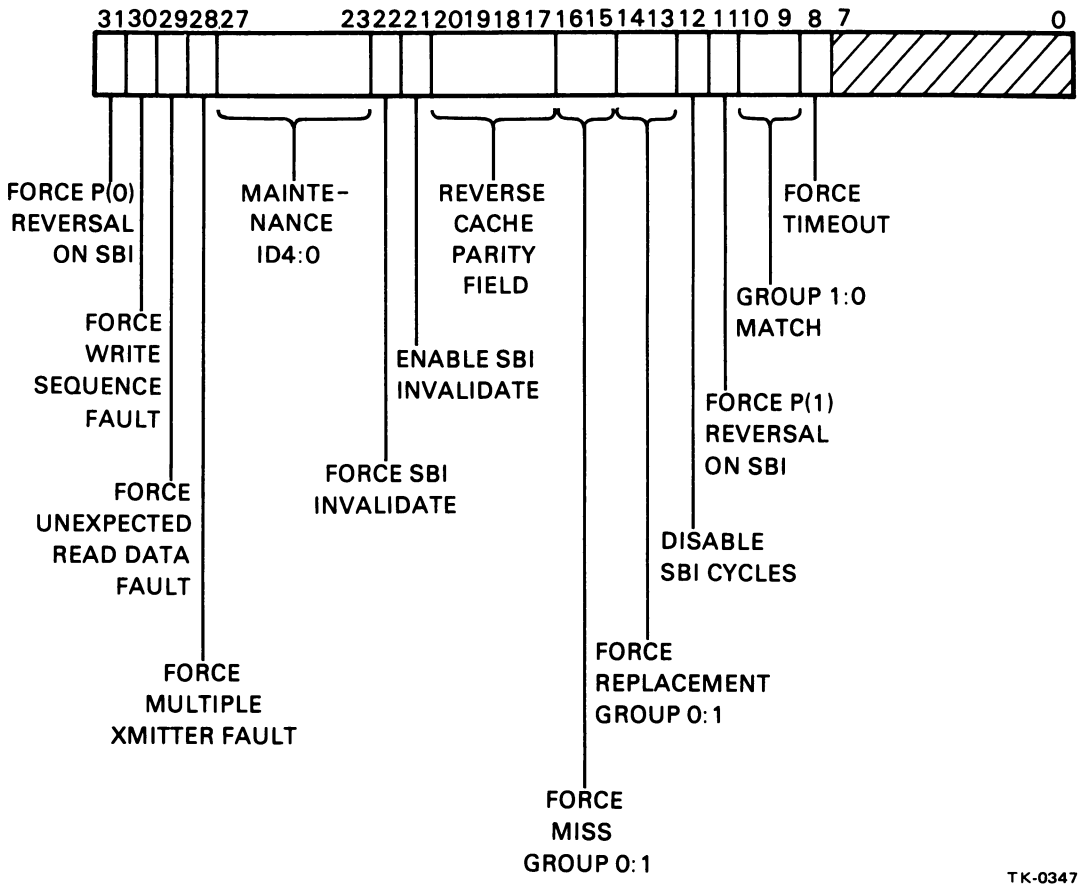


Figure 2-56 Maintenance Register

Table 2-14 Maintenance Register

Bit	Function	Description
31	Force P(0) Reversal on SBI	If set, the appropriate parity generator in the SBI interface is reversed. The fault does not occur until the interface transmits information to the SBI. This bit is read/write.
30	Force Write Sequence Fault	With this bit set, all writes by the SBI Control result in a write sequence fault. This is accomplished by changing the Write Data tag to the tag reserved for diagnostic use. This bit is read/write.
29	Force Unexpected Read Data Fault	With this bit set, the SBI Control transmits the following information format: Tag = Read Data (000) ID = Maintenance ID Data = undefined. The information is transmitted with good parity and will result in an unexpected read data fault in the nexus selected by the maintenance ID. This bit is read/write.
28	Force Multiple Transmitter Fault	With this bit set, a multiple transmitter fault can be forced in any nexus. For any nexus other than the CPU, the fault is forced by reading its configuration register. Once the Command/Address specifying the read has been transmitted, the following information format is transmitted by the CPU: Tag = 111 (reserved tag) ID = Maintenance ID Data = undefined. When the nexus transmits the read data (with ID = CPU ID), a multiple transmitter fault occurs, provided the maintenance ID was set to a value other than that of the CPU. A multiple transmitter fault is forced in the CPU by executing a write command in which the maintenance ID is transmitted with the write data. When the received ID is compared against the CPU ID, the ID mismatch results in a multiple transmitter fault. This bit is read/write.
27:23	Maintenance ID (4:0)	These bits are used to force unexpected read data faults, multiple transmitter faults, and as a compare field for the SBI Silo Comparator register. These bits are read/write.

Table 2-14 Maintenance Register (Cont)

Bit	Function	Description
22	Force SBI Invalidate	If set, any write executed by the CPU on the SBI becomes a write invalidate to Cache. This bit is read/write.
21	Enable SBI Invalidate	If set, write invalidates from the SBI are allowed (normal system operation). If not set, write invalidates are ignored. This bit is read/write.
20:17	Reverse Cache Parity Field	With this field set to one of the following codes, a parity error will occur in the selected byte when a Cache location is indexed. This can only occur if the selected byte has odd parity.

Bits 20	19	18	17	Group	Adrs/Data Byte
0	0	0	0	-	-
0	0	0	1	1	A1
0	0	1	0	1	A2
0	0	1	1	1	A3
0	1	0	0	0	A1
0	1	0	1	0	A2
0	1	1	0	0	A3
0	1	1	1	-	-
1	0	0	0	1	D3
1	0	0	1	1	D2
1	0	1	0	1	D1
1	0	1	1	1	D0
1	1	0	0	0	D3
1	1	0	1	0	D2
1	1	1	0	0	D1
1	1	1	1	0	D0

To force a parity error trap, the appropriate Cache operation must be initiated. These bits are read/write.

Table 2-14 Maintenance Register (Cont)

<table>
<tr><th>Bit</th><th>Function</th><th>Description</th></tr>
<tr><td>16, 15</td><td>Force Miss Group 0:1</td><td>When these bits are set to one of the following codes, a Cache miss is forced:

<table><tr><th>Bits
16</th><th>15</th><th>Function</th></tr><tr><td>0</td><td>0</td><td>No misses forced</td></tr><tr><td>0</td><td>1</td><td>Force miss on Group 1</td></tr><tr><td>1</td><td>0</td><td>Force miss on Group 0</td></tr><tr><td>1</td><td>1</td><td>Force miss on Group 1 and 0.</td></tr></table>
Forced misses are only permitted during read requests by the data path or instruction buffer. Forced misses are ignored for any write or invalidate operations (Cache must always contain the most current data). Parity errors are ignored during a forced miss. These bits are read/write.</td></tr>
<tr><td>14, 13</td><td>Force Replacement Group 0:1</td><td>Group selection for replacement is normally random in Cache. These bits override the random bit and select replacement as follows:

<table><tr><th>Bits
14</th><th>13</th><th>Cache Replacement</th></tr><tr><td>0</td><td>0</td><td>Random</td></tr><tr><td>0</td><td>1</td><td>Group 1 always</td></tr><tr><td>1</td><td>0</td><td>Group 0 always</td></tr><tr><td>1</td><td>1</td><td>Undefined</td></tr></table></td></tr>
<tr><td>12</td><td>Disable SBI Cycles</td><td>If set, SBI cycles are inhibited. For read operations with a Cache miss, the data in the D register of the data path will be unpredictable. This bit is read/write.</td></tr>
</table>

Table 2-14 Maintenance Register (Cont)

Bit	Function	Description
10, 9	Group 1:0 Match	These bits indicate the status of the match signals during the last read reference. The code is interpreted as follows: **Bits 10 9 — Indication** 0 0 — No Tag Match (SBI cycle initiated) 0 1 — Match Indicated in Group 1 1 0 — Match Indicated in Group 0 1 1 — Does not occur under normal operation These bits are read-only.
8	Force Timeout	If set, enables a forced timeout during a read operation. To generate the timeout, the timeout counter is loaded with the value FF after acknowledge of the read command is received. This bit is read/write.

Figure 2-57 illustrates the decision logic. As seen in this figure, the SBI cycle may be inhibited for any of the following:

1. The SBI Control is busy executing a previous SBI request (SBLN BUFFER FULL FF L is asserted).

2. The SBI Control is using the PA and MD buses during this cycle to either transfer read data to the instruction buffer or execute an I/O write invalidate cycle (Paragraph 2.3.2.11). (SBLS SBI CYCLE LTH H is asserted.)

3. The Translation Buffer determines the SBI cycle should be aborted (TBMU CANCEL L is asserted).

4. A maintenance function (SBLN DISABLE SBI CYC H is asserted).

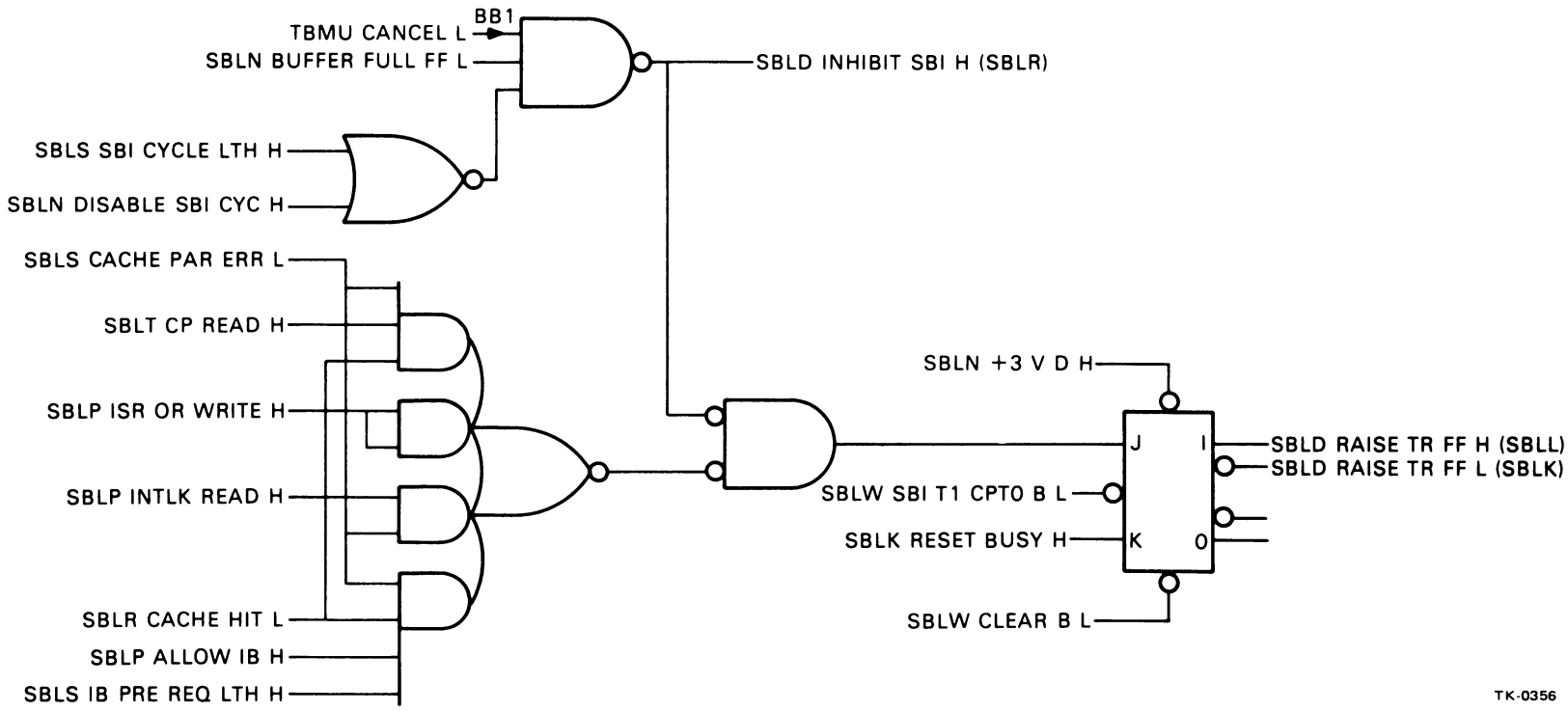


Figure 2-57 Start SBI Cycle Logic

2.3.2.5 State Generator Logic - The state generator provides timing pulses to decode confirmation and ISR responses from the SBI. Figure 2-58 illustrates the associated logic.

Initially, the S0 and S1 inputs are low for the assertion of a transfer request. This holds the state generator, which is basically a shift register, in an initial state (contents all 0s).

When ARB OK is returned, a command/address is transmitted generating SBLL TRANSMIT CA H. This signal generates SBLE START PULSE H and SBLK BUSY L at CPT0. SBLK BUSY L enables the S0 input which shifts the start pulse to generate the first timing pulse. This pulse, SBLL TIMING PULSE 0 H, is generated at CPT1. When SBLU ARB OK L is dropped, SBLE START PULSE H is negated at the following CPT0. This causes SBLL TIMING PULSE 0 H to be negated at CPT1. Coincidentally, SBLL TIMING PULSE 1 H is generated. Each subsequent CPT1 shifts the pulse to the next output. The resulting timing pulses are shown in Figure 2-59.

The timing pulses continue to be generated until:

1. A retry is initiated on the SBI (i.e., the Command/Address receives BSY or NR, or the Write Data receives BSY, NR, or ERR).

2. The last acknowledgment of the operation is received, an error confirmation is received, or a timeout occurs.

If a retry is initiated on the SBI, SBLK SET RETRY H is asserted and, at the following CPT0, SBLL RETRY FF L is generated. With SBLK BUSY L still asserted, the shift register is loaded with all 0s. SBLL RETRY FF L remains asserted until the next transmission on the SBI or until BUSY is reset.

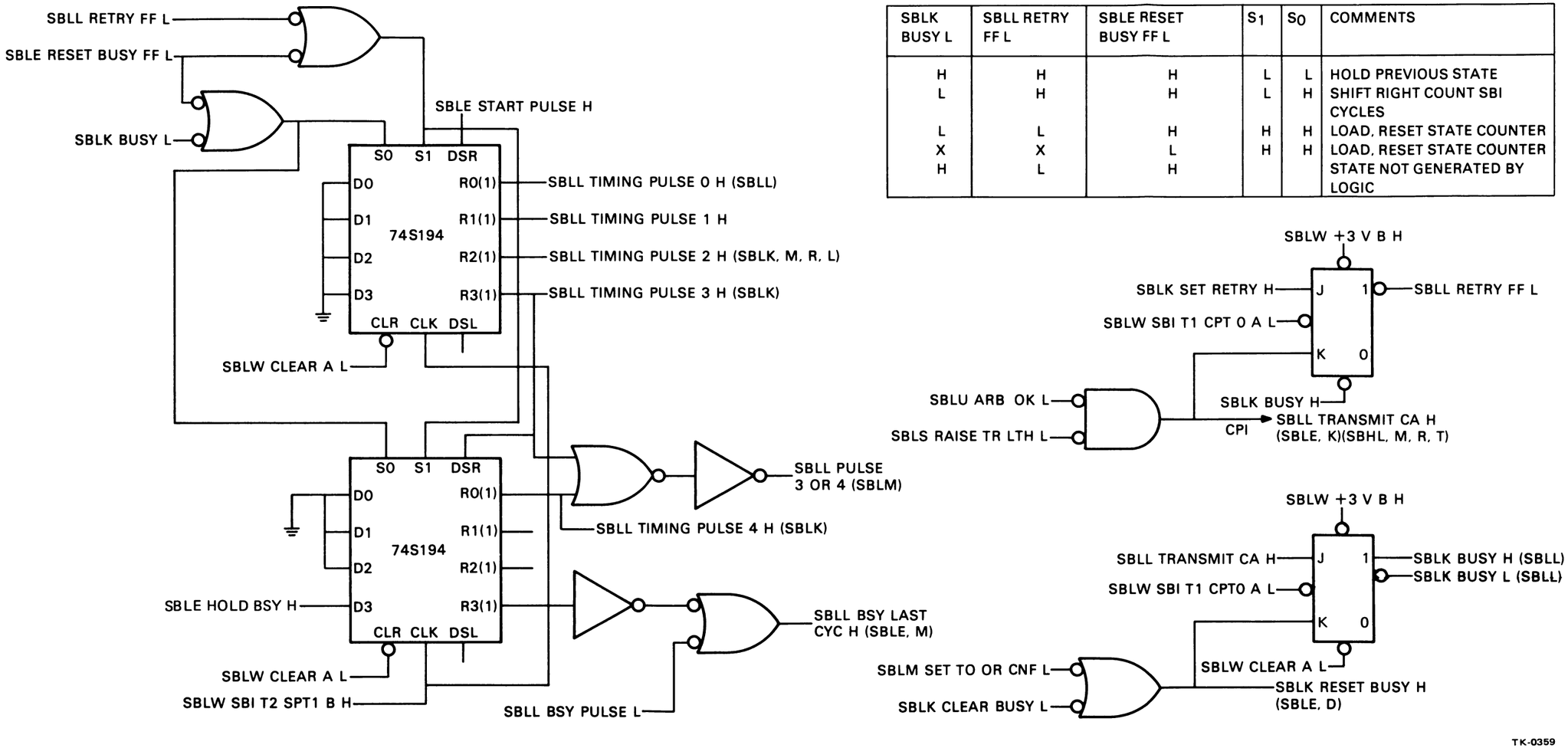
The state generator is also cleared when the last acknowledgment of an SBI operation is received. When this occurs, SBLK CLEAR BUSY L is generated by the State ROM (described next). This signal asserts SBLK RESET BUSY H to clear SBLK BUSY L and the state generator. SBLK RESET BUSY H is also generated when a timeout occurs (SBLM SET TO OR CNF L asserted).

The State Control ROM which generates SBLK CLEAR BUSY L is located on sheet SBLK of the engineering print set. This ROM is also responsible for generating SBLK SET RETRY H.

2.3.2.6 Expect Read Data - During a read operation, the Expect Read Data flag is set so that a timeout can be generated if the read data does not return. The Expect Read Data flag is also used as part of SBI protocol to set fault when read data is received but not expected. The logic is shown in Figure 2-60. As seen in this Figure, the Expect Read Data flag is the output of a shift register. As long as SBLK EXPECT RD H is set, SBLK BUFFER FULL remains asserted. SBLK BUFFER FULL is used during a read to hold the address for Cache, prohibit the initiation of another SBI cycle, and remember if the Read Data is for the IB or data path.

The Expect Read Data flag (SBLK EXPECT RD H) is set when acknowledge (ACK) is received for a read command. This flag is set for two decrements if the read command was an Extended Read. (The Extended Read operation is used for all reads by the SBI Control except I/O and interlock.) Similarly, Read Masked and Interlock Read Masked commands set the flag for one decrement.

Each time a read data format is decoded, BHM ANY READ DATA L is generated and the Expect Read Data flag is decremented. When the flag has cleared, SBLK BUFFER FULL is removed. The Expect Read Data flag is unconditionally cleared when a timeout occurs on the SBI (SBLM SET TO OR CNF L is asserted). The flag is also cleared when UNJAM is asserted on the SBI (SBLP FORCE HOLD L is generated).



SBLK BUSY L	SBLL RETRY FF L	SBLE RESET BUSY FF L	S_1	S_0	COMMENTS
H	H	H	L	L	HOLD PREVIOUS STATE
L	H	H	L	H	SHIFT RIGHT COUNT SBI CYCLES
L	L	H	H	H	LOAD, RESET STATE COUNTER
X	X	L	H	H	LOAD, RESET STATE COUNTER
H	L	H			STATE NOT GENERATED BY LOGIC

Figure 2-58 State Generator Logic

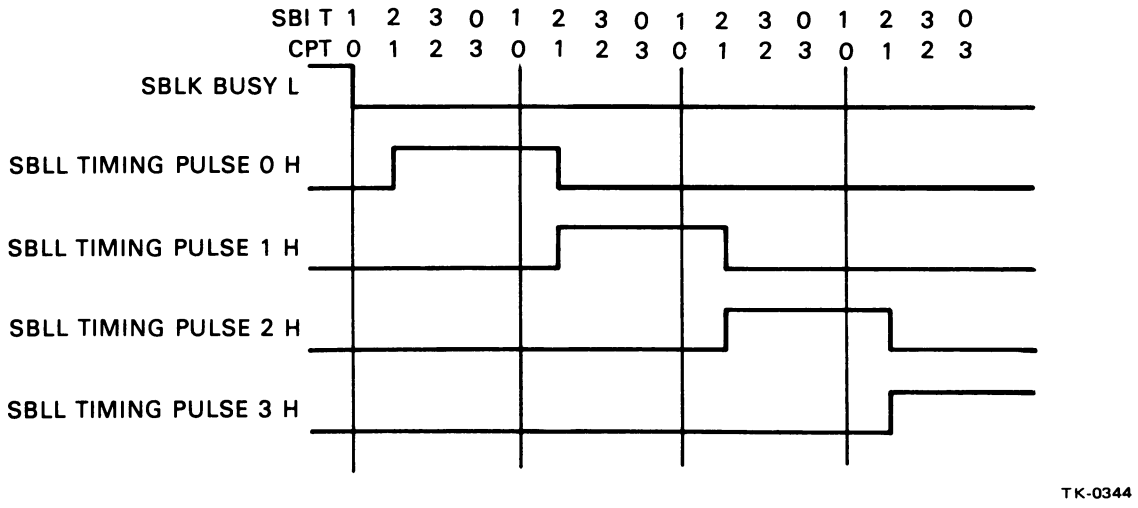
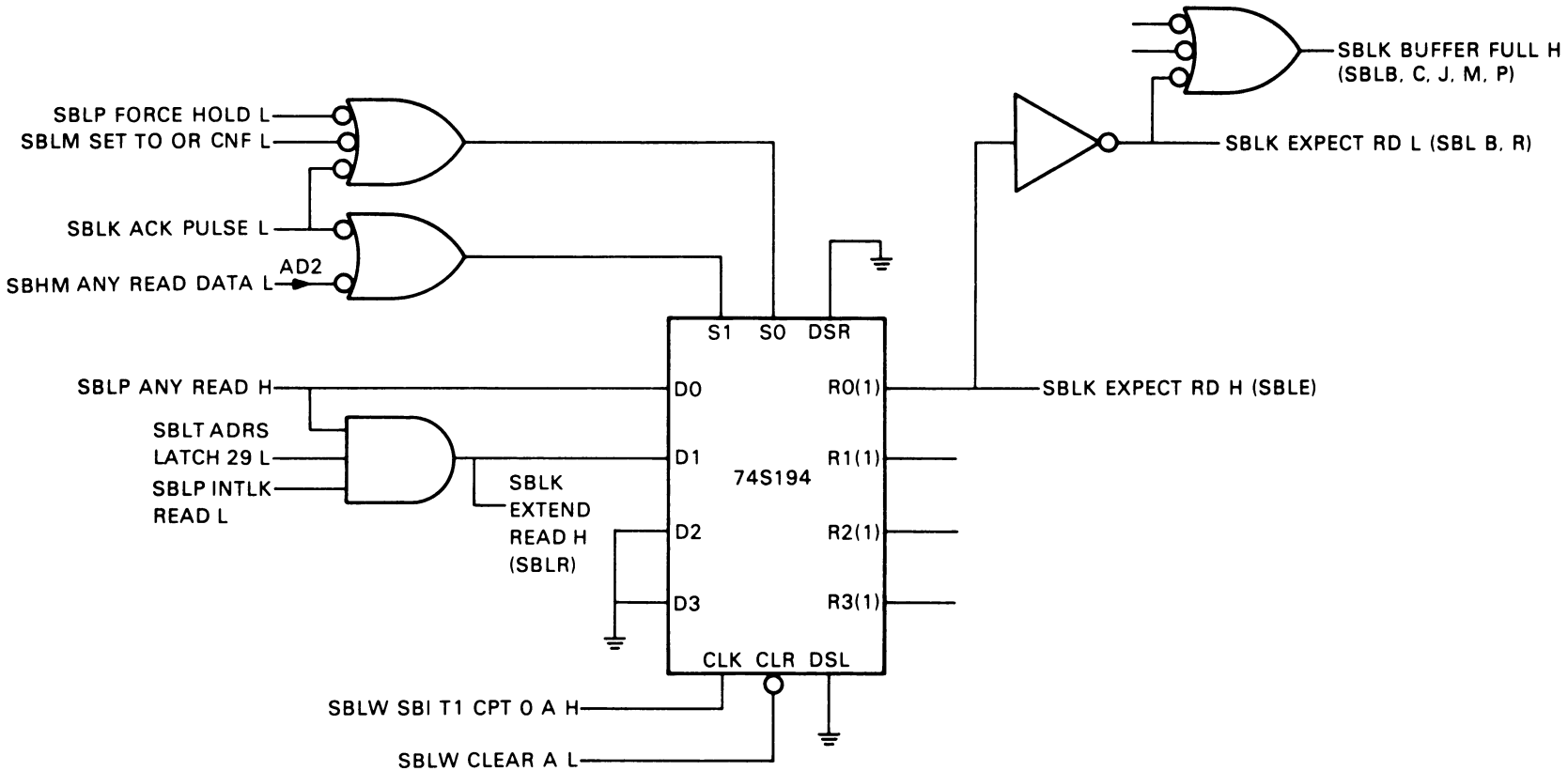


Figure 2-59 State Generator Timing Pulses



D_0	D_1	S_1	S_0	FUNCTION	COMMENTS
L	L	H	H	LOAD	CPU DOES NOT EXPECT ANY DATA
L	H	H	H		STATE NOT GENERATED BY LOGIC
H	L	H	H	LOAD	CPU EXPECTS 1 LONGWORD OF DATA
H	H	H	H	LOAD	CPU EXPECTS 2 LONGWORDS OF DATA
X	X	L	L	HOLD	NO CHANGE
X	X	L	H	SHIFT RIGHT	CLEAR EXPECT READ DATA
X	X	H	L	SHIFT LEFT	SUBTRACT 1 FROM THE AMOUNT OF LONGWORDS EXPECTED BY THE CPU

TK-0357

Figure 2-60 Expect Read Data Logic

2.3.2.7 Timeout Counter – Two types of timeouts may occur on the SBI. The first type of timeout limits the time it takes for the SBI Control to receive acknowledgment (ACK) for the transmission of a command. The second type is associated with read commands. This timeout limits the time between acknowledgment of a read command and the reception of the requested data. Both timeouts provide a time limit of 512 SBI cycles.

Figure 2-61 illustrates the timeout counter of the SBI Control. As seen in this Figure, SBLN BUFFER FULL H is generated when a transfer request is asserted by the SBI Control. On write commands this signal is cleared at the same time SBLK RESET BUSY L is generated. This resets the counter because SBLE BUFFER FULL FF L is connected to the counter clear line. Each subsequent CPT1 clocks the counter until acknowledgment (ACK) of the transmitted command is received. When acknowledge is received, SBLK CLEAR BUSY L is asserted by the state generator to generate SBLK RESET BUSY H. This signal is used to assert SBLE RESET BUSY FF H which initializes the counter by presetting it to 0.

If the command was a read command, the counter is again incremented until the read data is received. SBLE READ DATA FF L is generated for each read data format received. The return of the last read data clears SBLN BUFFER FULL FF H to clear the counter.

A timeout occurs in either case if the counter is permitted to overflow. If acknowledgment does not return within 512 cycles for the transmission of a command, SBLK TIMEOUT CARRY H is asserted and a timeout occurs. This asserts SBLK BUFFER FULL and aborts the cycle. Similarly, should read data not be returned within 512 SBI cycles after acknowledge is received, SBLK TIMEOUT CARRY H is asserted and a timeout occurs. This resets SBLK BUFFER FULL and clears the counter.

2.3.2.8 STALL Signal Logic – Whenever a Cache read miss occurs, the requested data must be fetched from main memory by the SBI Control. In accommodation, the SBI Control generates a stall signal (SBLT STALL L) and sends it to the microsequencer to delay CPU operation. This signal temporarily prevents the execution of the next microinstruction until the data is fetched.

The CPU is also stalled for a number of other reasons. The decision logic is provided on SBLT of the engineering print set. Sheets 24 and 25 of the print set provide an explanation of the conditions. The basic cases in which stall is generated and cleared are listed in Table 2-15. As seen from the examples listed, a STALL is asserted because the requested Cache operation cannot be executed at that time. During the stall, the CPU waits until the operation can be done.

NOTE

A stall can only be generated during requests by the data path. Stall is never generated for an IB reference.

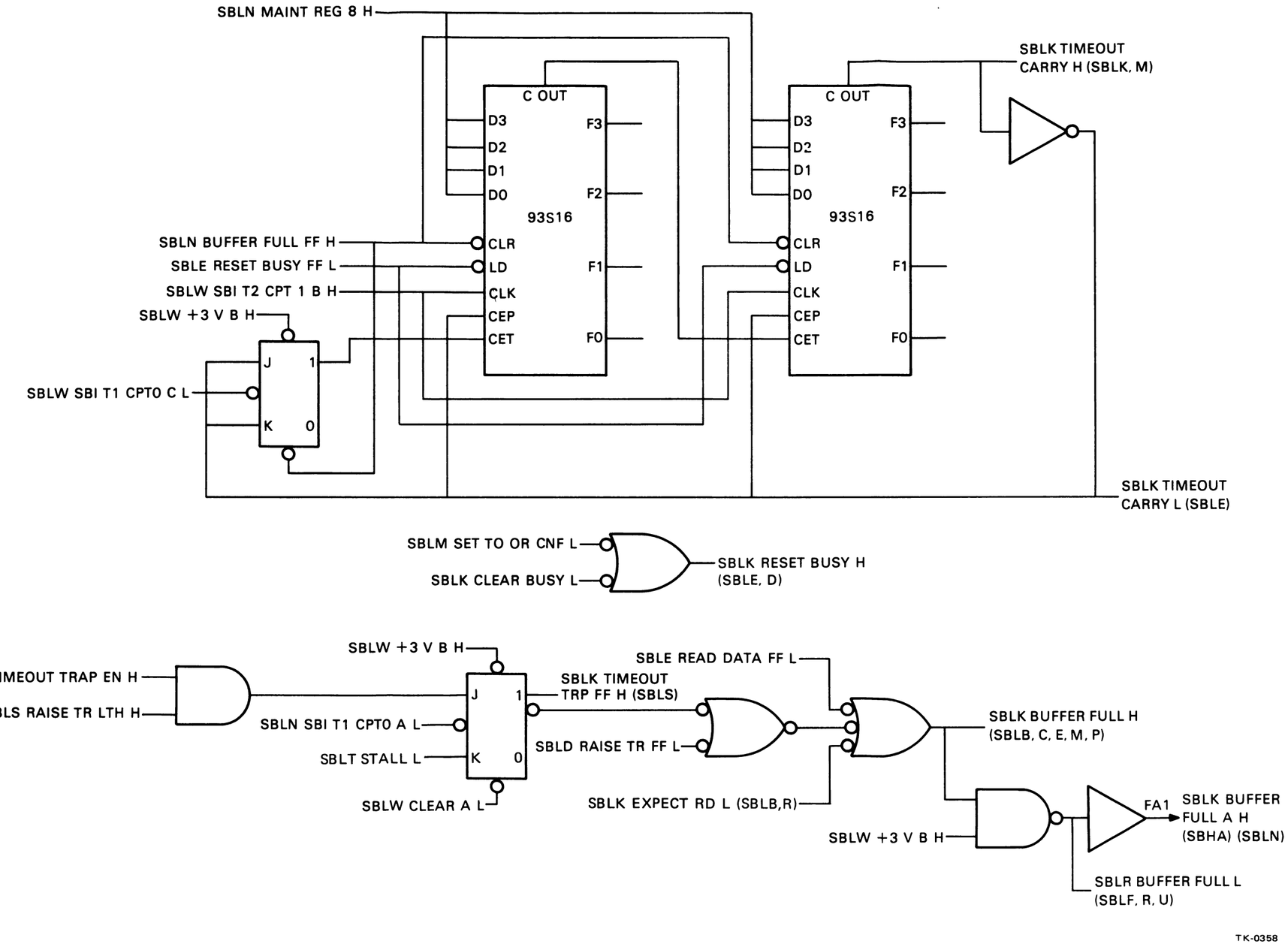


Figure 2-61 Timeout Counter Logic

Table 2-15 Stall Conditions

Condition Which Generates STALL	Condition Which Clears STALL
The VA mux selects the VIBA register instead of the VA register during an auto-reload of the IPA register.	The VA mux selects the VA register.
The SBI Control is using the PA or MD bus for: 1. An I/O Write Invalidate cycle 2. For transfer of read data to Cache.	The CPU executes an operation over the PA and MD buses.
A Cache read miss occurs on a reference by the data path.	The requested read data is fetched from memory, an error confirmation is returned, or a timeout occurs.
The data path requests the SBI for a write and the SBI Control's write buffer is full.	The SBI Control's write buffer is available.
An ISR is issued by the CPU.	A data response is returned to the CPU or a timeout occurs.
A write to I/O space is issued by the CPU.	Acknowledgment (ACK) is received by the CPU for the write data, an error confirmation is returned, or a timeout occurs.

2.3.2.9 Cache Valid Bit Logic – A Cache entry is marked invalid by enabling the Cache write pulses while the valid bit input to Cache (SBLR VALID H) is held low. The Cache valid bit input is dropped for any of the following.

1. The first of two longwords during a Cache update.
2. An I/O Write to main memory (Paragraph 2.3.2.11).
3. An I/O Write to main memory during a CPU read to the same address (Paragraph 2.3.2.11.1)
4. An explicit Invalidate cycle by the MCT microfield.
5. An Interlock Read Masked operation by the CPU.
6. An Extended Write Masked operation by the CPU. (The CPU uses Extended Write Masked operations only when it intends to clear parity errors in memory.)

The selection of an Invalidate cycle (case 4) unconditionally executes a dedicated cycle to mark a Cache entry invalid. This cycle is selected by the MCT microfield for clearing Cache by microcode during power up and for diagnostic purposes. Similarly, an I/O Write to main memory (step 2) initiates a dedicated cycle to mark the Cache entry (if in Cache) invalid. This cycle (called I/O Write Invalidate) is initiated by the SBI Control when the condition occurs.

Figure 2-62 illustrates the logic associated with setting or clearing the valid bit input.

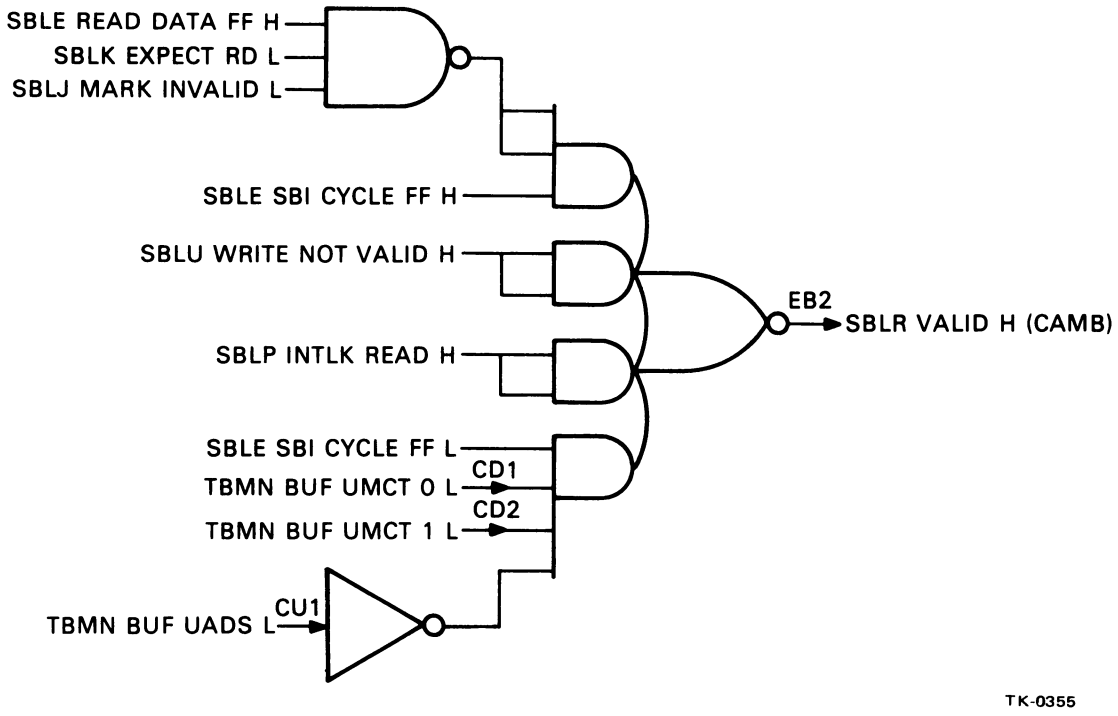


Figure 2-62 Cache Valid Bit Input Logic

For the case of the first of two longwords during a Cache update, SBLK EXPECT RD L remains low for the first longword and is negated for the second. This ensures the first longword is always marked as invalid. SBLE READ DATA FF H, SBLJ MARK INVALID L, and SBLE SBI CYCLE FF H are high for both longwords.

When an I/O write to memory is detected during a CPU read to the same location, SBLJ MARK INVALID L is generated to mark the returning data invalid. During any I/O write to main memory, SBLE SBI CYCLE FF H is set and SBLE READ DATA FF H remains unasserted to drop the valid bit input.

SBLU WRITE NOT VALID H is asserted to invalidate an entry when an explicit Invalidate cycle is initiated by the microcode. This signal is also generated when a Cache parity error occurs during a write (Paragraph 2.3.2.10).

SBLP INTLK READ H is asserted to invalidate the returning data for Cache when an Interlock Read Masked operation is executed by the CPU.

Extended Write Masked operations are used by the CPU only when it intends to clear parity errors in memory. As seen in Figure 2-62, this operation is selected under microcode control.

2.3.2.10 Cache Parity Errors During Writes – If a parity error is detected in a Cache tag during a write hit, a miss occurs and the write pulses are inhibited. The entry is then marked invalid to avoid the problem of stale data in Cache. This is accomplished by the SBI Control which:

1. Presents the address to Cache
2. Drops the valid bit input.
3. Enables write pulses to the address matrix.

The associated logic is shown in Figure 2-63. A discussion of the logic is provided next.

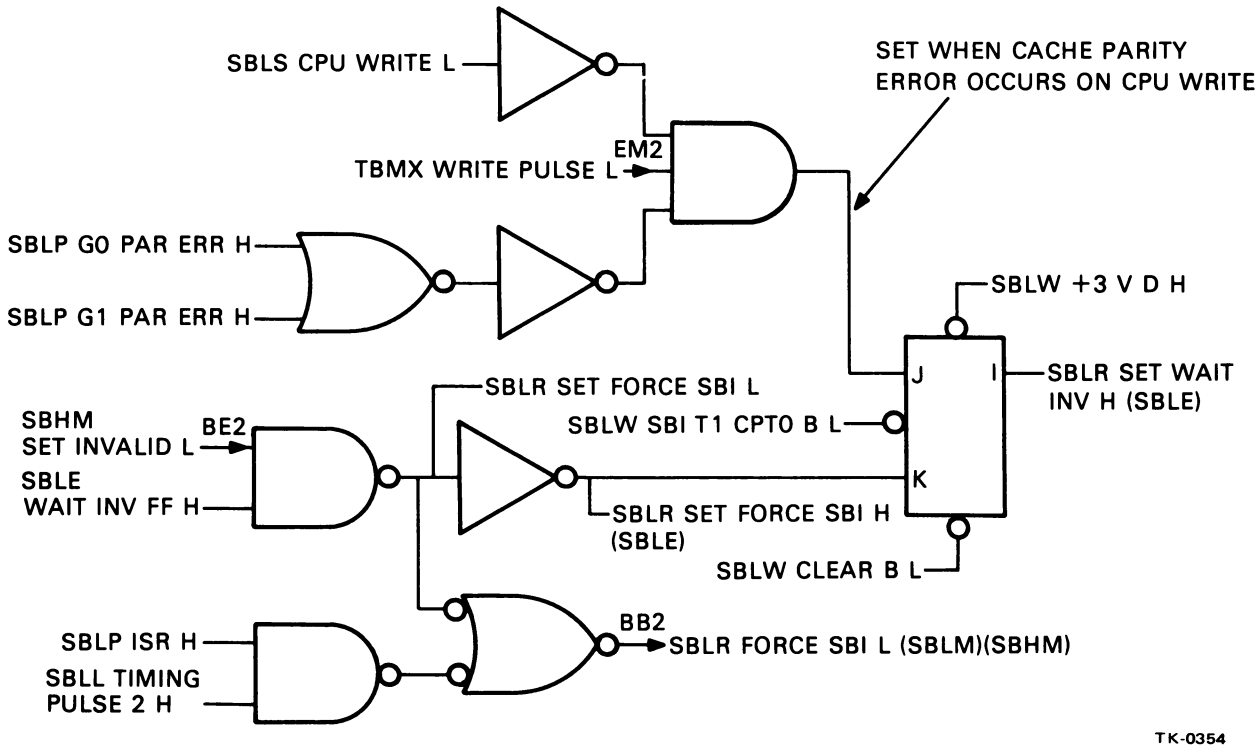


Figure 2-63 Write Parity Error Logic

The address is presented to Cache by enabling the contents of the address register to the PA bus. SBLR SET WAIT INV H is generated and sent to SBLE for deskewing. The result, SBLE WAIT INV FF L, is used to generate SBLR SET FORCE SBI L providing an I/O Write Invalidate cycle is not in progress. (I/O Write Invalidate cycles have priority on the PA bus, paragraph 2.3.2.11.) SBLR SET FORCE SBI L asserts SBLR FORCE SBI L to gain control of the MD and PA buses. This signal generates SBHM SELECT SBI ADR L which enables the address mux to select the address register for the PA bus.

With the address available to Cache, SBLR FORCE SBI L is also deskewed on SBLE to generate SBLE FORCE SBI FF L. This signal asserts SBLU WRITE NOT VALID H. SBLU WRITE NOT VALID H is one of the conditions which negate the valid bit, SBLR VALID H. With SBLR VALID H negated, the valid bit input to the indexed Cache location is dropped (Paragraph 2.3.2.9).

With the Cache location indexed and the valid bit input negated, the Cache write pulses can be enabled. For this SBLU WRITE NOT VALID H generates SBLN SBI MISS DATA G0 H and SBLN SBI MISS DATA G1 H. These signals enable write pulses to both groups of the address matrix whether or not a Cache hit occurs.

2.3.2.11 I/O Writes to Memory - If any nexus other than the CPU executes a write to a memory location which is also contained in Cache, the Cache entry is invalidated. This is done to avoid the problem of stale data in Cache. (Stale data is defined as a valid Cache entry which differs from main memory.) Each time an I/O write command is sent to memory, the address is also latched by the SBI Control and presented to Cache via the PA bus. This is accomplished by an I/O Write Invalidate cycle. If a Cache hit occurs, a write pulse is generated and the Cache entry becomes invalidated.

Figure 2-64 contains the associated decision logic. A discussion of the logic is provided below.

The SBI Control monitors the SBI for I/O writes to memory. SBHN BUS WRITE COM H is generated as a result of function decoding on SBHN. This signal indicates a write command/address is being transferred on the SBI. As seen in the Figure, this signal is ANDed with SBHD RECEIVE DATA 27 H which remains low for addresses of main memory. In addition, SBHM MY ID H remains low for commands initiated by a nexus other than the CPU. As a result, when a write is executed to main memory (not I/O space) by an I/O device, SBHM SET INVALID L is generated.

SBHM SET INVALID L is asserted to initiate an I/O Write Invalidate cycle. This signal generates SBHM WRITE INVALID L and SBHM SELECT SBI ADR L for the address mux. The assertion of these two signals selects the contents of the Read Data register for output to the PA bus. The Read Data register, which is loaded with SBI information every cycle, contains an address during this cycle. This presents the address to Cache for the invalidation of the Cache entry.

If Cache contains a valid entry corresponding to the address, a write pulse is generated in the appropriate group. With the valid bit input (SBLR VALID H) held low, the Cache entry is marked invalid.

2.3.2.11.1 I/O Write to Memory (Special Case) - The SBI Control also monitors the SBI for an I/O write during a CPU read to the same address. This is necessary because of memory's command buffer. Figure 2-65 illustrates the problem. An explanation follows.

Under normal operation, when memory is busy, all incoming commands are stored in memory's buffer. For this example assume memory is busy and a read command by the CPU is stored in the buffer. A write command to the same address is then issued by an I/O device. The write command along with the write data is likewise stored in the buffer. The transfer of the I/O write command is detected by the monitoring SBI Control which initiates an I/O Write Invalidate cycle (Paragraph 2.3.2.11). Cache, however, does not contain the entry at this time, so no write pulses are generated.

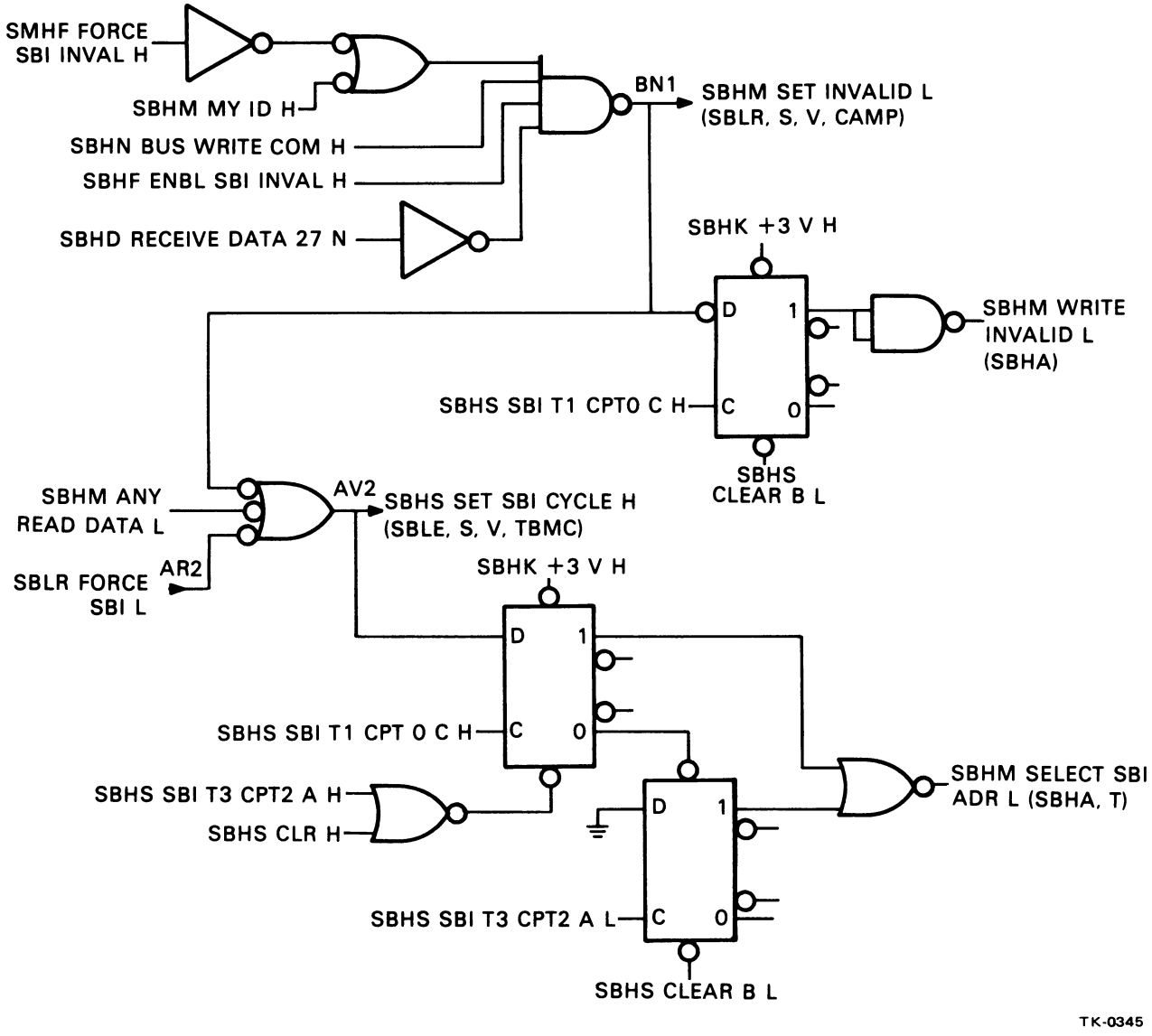


Figure 2-64 I/O Write Invalidate Logic

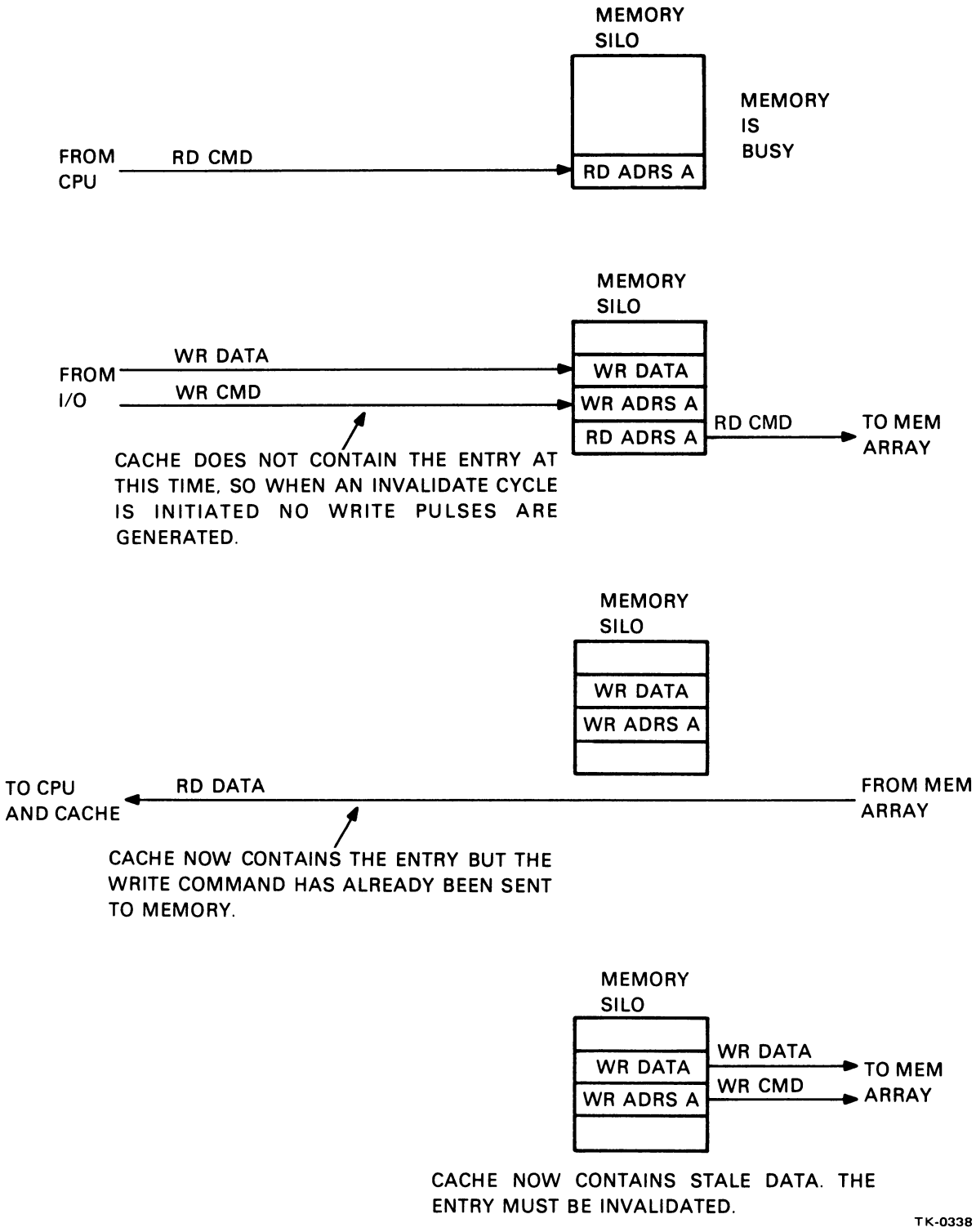


Figure 2-65 Case of I/O Write to an Address Being Read by the CPU

Eventually, memory becomes available and the read command is removed from the silo for processing by memory. The requested data is then fetched from the array and transferred to the CPU for use and also for storage in Cache (normal Cache operation).

With the read operation complete, memory removes the write command from the buffer and executes a write to the same memory location. This results in a stale data situation.

To avoid this situation of stale data in Cache, the SBI Control monitors the SBI for the circumstances described. When these circumstances are detected, the Cache entry is marked invalid. Figure 2-66 illustrates the associated logic and a discussion is provided next.

The logic is primarily implemented with a comparator which compares the SBI address from the Read Data register with the address of the address register. This occurs when an I/O write to main memory is executed. (SBHM SET INVALID L is set when an I/O write to main memory is detected, paragraph 2.3.2.9.) Only a portion of the addresses are compared. If the addresses match, SBLJ MARK INVALID L is generated. This indicates the situation described has occurred. SBLJ MARK INVALID L holds the valid bit input low. As a result, the read data is marked invalid when it returns from memory.

2.3.3 Memory Control Functions

The CPU initiates various memory functions during normal operations. CPU-initiated memory functions are selected by part of the microword. CS bits (47:42) are used to control the Translation Buffer (TB) and SBI Control during a CPU memory access or internal register access (via the ID bus). The CS bus receivers are located in the TB. Copies of the signals received are sent to the SBI Control (SBH and SBL) (Figure 2-67).

Figure 2-68 shows the memory control microcode fields. As seen in this Figure, CS bits (46:43) provide control for both the MD bus and ID bus, depending on the condition of CS bit 42. The MD bus control is described next. For a description of the ID bus control, refer to the Data Path Technical Description (EK-KA780-TD-PRE).

When CS bit 42 equals 0, CS bits (46:43) define the UMCT field (memory control field). This field, together with the ADS field, selects a memory function. Table 2-16 lists the various memory control functions as selected by the CS bits and explains each function.

NOTE

Although not all functions perform a memory operation, the functions are referenced as memory control functions because they are selected by the memory control field of the microword.

In Table 2-16, the column labled INTENDED ACCESS CHECK actually refers to page accessibility. This check determines if a read or write access is allowed during the current processor mode. The protection check (and M bit check) is executed only during functions which include address translations.

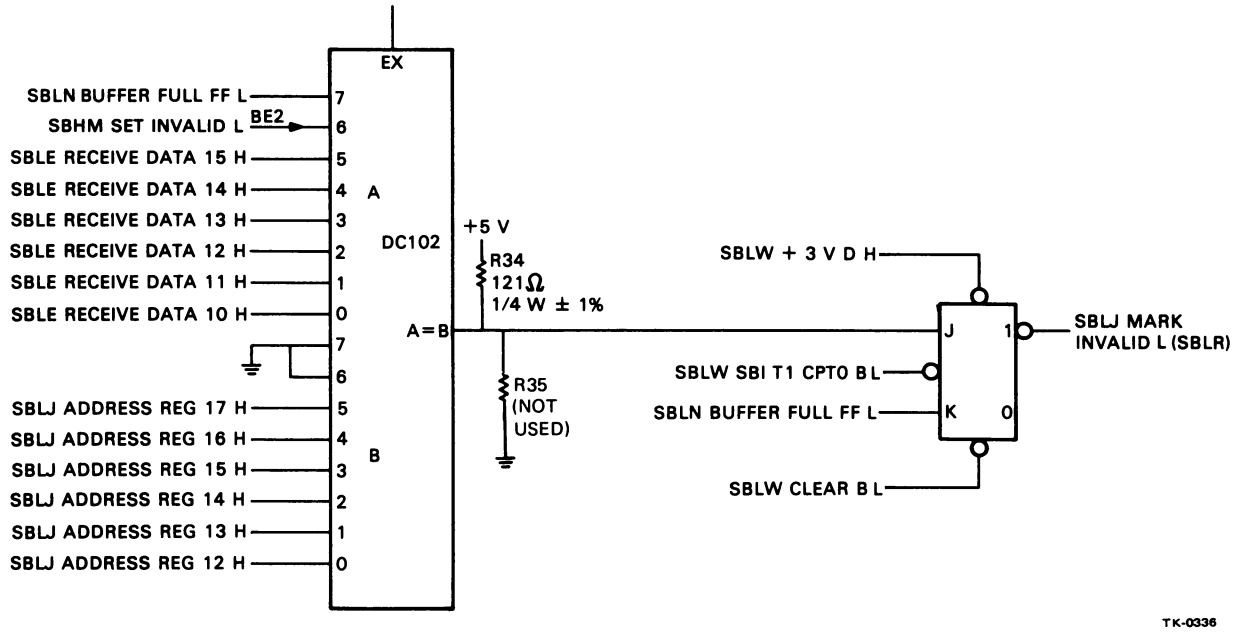


Figure 2-66 I/O Write Detection Logic (Special Case)

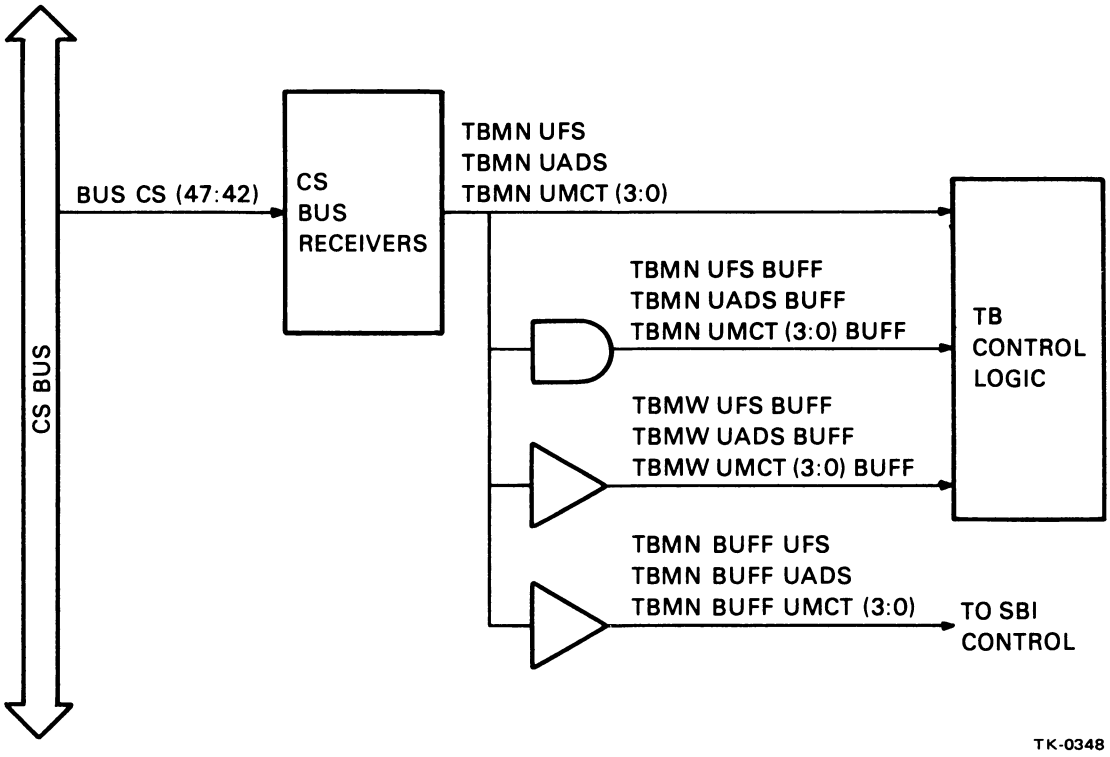
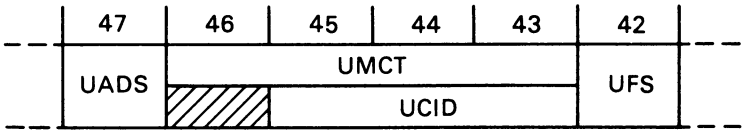


Figure 2-67 Control via the CS Bus



UADS – SELECTS A VIRTUAL OR PHYSICAL MEMORY REFERENCE (0 = VIRTUAL, 1 = PHYSICAL)

UFS – SELECTS THE INTERPRETATION OF CS <46:43>
0 = UMCT – USED FOR MD BUS CONTROL DURING MOST MEMORY OPERATIONS
1 = UCID – USED FOR ID BUS CONTROL

UMCT – MEMORY CONTROL FIELD

UCID – CONSOLE AND ID BUS CONTROL FIELD

TK-0337

Figure 2-68 Memory Control Microcode Fields

Table 2-16 Microcode Selected Memory Control Functions

Microfield										
	MCT					Memory Function Mnemonic	Address Type	Intended Access Check	SBI Command	Retryable	Comment
ADS	3	2	1	0	FS						
0	0	0	0	0	0	TEST.RCHK	Virtual	For Read (No Trap)	None	No	See Note 1
0	0	0	0	1	0	MEN.NOP	Virtual		None	No	
0	0	0	1	0	0	TEST.WCHK	Virtual	For Write (No Trap)	None	No	See Note 1
0	0	0	1	1	0	RESERVED					
0	0	1	0	0	0	RESERVED					
0	0	1	0	1	0	WRITE.V.NOCHK	Virtual	None	Write Masked	No	See Note 2
0	0	1	1	0	0	WRITE.V.WCHK	Virtual	For Write	Write Masked	Yes	
0	0	1	1	1	0	LOCKWRITE.V.XCHK	Virtual	Unspecified	Interlock Write Masked	No	
0	1	0	0	0	0	READ.V.RCHK	Virtual	For Read	Extended Read	Yes	See Note 3
0	1	0	0	1	0	READ.V.NOCHK	Virtual	None	Extended Read	No	See Note 2 & 3
0	1	0	1	0	0	READ.V.WCHK	Virtual	For Write	Extended Read	Yes	See Note 3
0	1	0	1	1	0	READ.V.IBCHK	Virtual	For Read or Write	Extended Read	Yes	See Note 3 & 4
0	1	1	0	0	0	READ.V.NEWPC	Virtual	For Read (No Trap)	Extended Read	No	See Note 3 & 5
0	1	1	0	1	0	LOCKREAD.V.NOCHK	Virtual	None	Interlock Read Masked	No	See Note 6
0	1	1	1	0	0	LOCKREAD.V.WCHK	Virtual	For Write	Interlock Read Masked	Yes	See Note 6
0	1	1	1	1	0	RESERVED					
1	0	0	0	0	0	SBI.HOLD			Assert Hold	No	
1	0	0	0	1	0	SBI.HOLD+UNJAM			Assert Hold and Unjam	No	
1	0	0	1	0	0	INVALIDATE	Physical		None	No	See Note 7
1	0	0	1	1	0	VALIDATE	Physical		None	No	See Note 8
1	0	1	0	0	0	EXT WRITE.P	Physical		Extended Write Masked	No	See Note 9
1	0	1	0	1	0	WRITE.P	Physical		Write Masked	No	See Note 10
1	0	1	1	0	0	RESERVED					
1	0	1	1	1	0	LOCKWRITE.P	Physical		Interlock Write Masked	No	
1	1	0	0	0	0	RESERVED					
1	1	0	0	1	0	READ.P	Physical		Extended Read	No	See Note 3 & 11
1	1	0	1	0	0	RESERVED					
1	1	0	1	1	0	READ.INT.SUM			Interrupt Summary Read	No	
1	1	1	0	0	0	RESERVED					
1	1	1	0	1	0	LOCKREAD.P	Physical		Interlock Read Masked	No	See Note 6
1	1	1	1	0	0	RESERVED					
1	1	1	1	1	0	ALLOW.IB.READ	Physical from IPA		Extended Read	No	See Note 3 & 12
0	X	X	X	X	1	NO MEMORY OPERATION			None	No	
1	X	X	X	X	1	DEFAULT: ALLOW.IB.READ	Physical from IPA		Extended Read	No	See Note 3 & 12

NOTES

1. These functions provide TB condition testing to prevent a microtrap within a microtrap. They enable branch code bits to the branch enable muxes from the TB. These conditions are then available if a branch enable 1D (Translation Test, BEN 1D) is performed in the next microinstruction. The branch code bits for each TB condition are listed here:

Branch Code 1	Branch Code 0	TB Condition
1	1	TB miss
1	0	Protection violation, not TB miss
0	1	M bit violation, not protection violation or TB miss
0	0	No problem

2. This function is used for cycles that are prechecked by microcode, such as writing page table entries.

3. This function initiates an Extended Read on the SBI only if a Cache miss occurs and the reference is not to I/O space. If the reference is to I/O space, a Read Masked is initiated.

4. During this function a protection read check or write check is performed as specified by the instruction buffer. This function is retryable as READ.V.RCHK or READ.V.WCHK.

5. During this function the read data obtained from memory is sent to the instruction buffer (IB). All errors are handled by the IB error circuitry. This function is used whenever the microcode wishes to reload the IPA register. The IPA must be reloaded when a macroprogram transfer of control occurs or to restart instruction prefetching after loading the TB with a translation of a previously missing page (required PTE).

6. During this function, a Cache miss is forced to initiate the SBI command.

7. This function writes a tag and data with good parity into the indexed location in both groups of Cache and marks these locations invalid by dropping the valid bit. This function is used when loading all Cache locations during power-up. It is also used for microdiagnostic purposes and certain error routines.

8. This function writes a tag and data into the indexed location in the Cache group specified by the normal replacement logic (usually the FORCE REPLACE bits in the Maintenance register on SBL) and then marks the location valid by setting the valid bit. This function is used for microdiagnostic purposes.

9. The data written during this function is unpredictable. This function is used to clear uncorrectable errors in main memory. The corresponding location in Cache, if any, is also invalidated when this function is executed.

10. This function is typically used during STPCTX.

11. This function is typically used during LDPCTX.

12. This function enables the initiation of read cycles by the instruction buffer (while performing the ID bus operation specified for one of the two codes.) It is used when an explicit memory operation is not required (i.e., I stream bytes are sent to the IB).

2.3.3.1 Retryable Memory Control Functions – A memory control function is retried by:

1. Performing a last reference microbranch (BEN11) to determine which retryable memory control function was just executed.

2. Asserting the memory control function again in combination with the proper saved context in the miscellaneous field (UMSC field).

When a retryable memory control function is to be executed, two code bits from the TB [TBMD LAST REF CODE (1:0) H] are enabled to the branch enable muxes. An indication of the selected memory control function is then available if a branch enable 11 (last reference microbranch, BEN11) is performed.

The retryable memory control functions are listed here with their corresponding code bits:

Memory Control Function	Last Reference Code	
	1	0
READ.V.RCHK	0	0
LOCK READ.V.WCHK	0	1
WRITE.V.WCHK	1	0
READ.V.WCHK	1	1

READ.V.IBCHK is also retryable as READ.V.RCHK or READ.V.WCHK.

2.3.3.2 Microtraps During Memory Control Functions – During the execution of a memory control function, a microtrap may occur. Table 2-17 lists the possible microtraps for each memory control function. The conditions for each of these microtraps are given below.

If a microtrap occurs during the execution of a memory control function, the reference is usually aborted. This is true for all microtraps except for the unaligned data microtrap and the Cache parity error microtrap. In the case of the unaligned data microtrap, the microtrap is executed as soon as all of the data of the aligned longword is accessed. For a Cache parity error microtrap, the reference is only aborted if it is a read reference. Otherwise, the function is executed regardless of the cache parity error.

TB Miss – A TB miss microtrap occurs when a requested page table entry is not found in the TB. During the TB Miss microtrap service routine, the PTE is fetched from main memory and placed in the TB.

Protection Violation – A protection violation microtrap occurs if the current processor mode and/or intended page access violates the assigned protection for the page as dictated by the protection code of the PTE.

Cross Page Boundary – A cross page boundary microtrap occurs when a cycle which crosses a page boundary is attempted. During the cross page boundary microtrap service routine, the intended access to the new page is checked before the cycle can be executed. This prevents the possibility of writing the first part of a data stream, after which the writing of the second part is prohibited (i.e., eliminates the possibility of half updated data).

Table 2-17 Microtraps During Memory Control Functions

Memory Control Function	Microtraps								
	TB Miss	Protection Violation	Cross Page Boundary	Unaligned Data	TB M Bit	Odd Address	TB Parity Error	Cache Parity Error	SBI Error
TEST.RCHK	N	N	N	N	N	N	Y	N	N
MEM.NOP	N	N	N	N	N	N	N	N	N
TEST.WCHK	N	N	N	N	N	N	Y	N	N
WRITE.V.NOCHK	Y	N	–	–	N	–	Y	N	Y
WRITE.V.WCHK	Y	Y	*	*	Y	Y	Y	N	Y
LOCKWRITE.V.XCHK	–	–	–	–	–	–	Y	N	Y
READ.V.RCHK	Y	Y	*	*	N	Y	Y	Y	Y
READ.V.NOCHK	Y	N	–	–	N	–	Y	Y	Y
READ.V.WCHK	Y	Y	*	*	Y	Y	Y	Y	Y
READ.V.IBCHK	Y	Y	Y	Y	Y	Y	Y	Y	Y
READ.V.NEWPC	N	N	N	N	N	Y	N	N	N
LOCKREAD.V.NOCHK	Y	N	–	–	N	–	Y	Y	Y
LOCKREAD.V.WCHK	Y	Y	–	–	Y	–	Y	Y	Y
SBI.HOLD	N	N	N	N	N	N	N	N	N
SBI.HOLD+UNJAM	N	N	N	N	N	N	N	N	N
INVALIDATE	N	N	N	N	N	N	N	N	N
VALIDATE	N	N	N	N	N	N	N	N	N
EXTWRITE.P	N	N	N	N	N	N	N	N	Y
WRITE.P	N	N	N	N	N	N	N	N	Y
LOCKWRITE.P	N	N	N	N	N	N	N	N	Y
READ.P	N	N	N	N	N	N	N	Y	Y
READ.INT.SUM	N	N	N	N	N	N	N	N	Y
LOCKREAD.P	N	N	N	N	N	N	N	Y	Y
ALLOW.IB.READ	N	N	N	N	N	N	N	N	N

N = Do not microtrap on condition,
– = Hardware behavior undefined; microcode must prevent condition.

Y = Microtrap on condition,
* = Microtrap on condition unless MSC/SECOND.REF. or RETRY.NO.TRAP.

Unaligned Data – An unaligned data microtrap occurs when a reference is across a longword boundary. During the microtrap service routine, the microcode retrieves the portion of the data which was not part of the original longword.

TB M Bit – A TB M bit microtrap occurs when a write is attempted to a page whose PTE contains an unasserted M bit. During the microtrap service routine, the M bit of the PTE is set in the TB and in memory. To accomplish this, the PTE in memory is fetched, modified, and rewritten.

Odd Address – An odd address microtrap occurs when a 16-bit reference is made to an odd byte boundary in compatibility mode. The microtrap service routine performs an abort.

TB Parity Error – A TB parity error microtrap occurs when a parity error is detected in the TB. The information from both groups of the address matrix and data matrix is parity checked as soon as an address is sent to the TB matrices.

Cache Parity Error – A Cache parity error microtrap occurs when a parity error is detected in Cache. The output of both groups of the address matrix and data matrix is parity checked as soon as a Cache reference is made.

SBI Error – An SBI error microtrap occurs when an SBI protocol error occurs.

2.3.4 Typical Write Timing

Figure 2-69 illustrates the timing of a typical CPU initiated Write Masked cycle. Note that the diagram is divided into CPU cycles rather than SBI cycles.

As seen in Figure 2-69 the write is initiated by part of the microword (memory control field) which controls the transfer of the address and data to Cache and the SBI Control. The Cache write pulse occurs at CPT 3. Note an SBI cycle must be initiated to update main memory whether or not a Cache hit occurs. This is true for all write cycles.

With SBLU ARB OK L low at the following CPT2, SBLK BUSY L is asserted. SBLU ARB OK L indicates no other nexus with higher priority has arbitrated for the SBI. SBK BUSY L begins the information transmission sequence [SBLL TIMING PULSE (0:3) H] by starting the state generator. When Acknowledge is received for the Write Data, SBLK RAISE TR FF H, SBLK BUFFER FULL H, and SBLK BUSY L are dropped indicating the cycle has ended.

2.3.5 Typical Read Miss Timing

Figure 2-70 illustrates the timing of a typical read miss in which an SBI cycle is executed to fetch the requested data for the data path and to update Cache. Note that the diagram is divided into CPU cycles rather than SBI cycles.

As seen in this Figure, the read is initiated by part of the microword. Without the generation of a Cache tag match signal, SBLL STALL L is generated to stall the CPU until the data can be retrieved from main memory. At CPT0 of the following cycle, SBLD RAISE TR FF H is asserted to start an SBI cycle. SBLK BUFFER FULL H is raised to lock the PA register and disable further latching from the PA bus. Using the address from the PA register, a command/address is transmitted to the SBI at T0. At SBI T1, SBLK BUSY L is asserted to start the state generator and remove the transfer request. SBLL TIMING PULSE (2:0) H are generated to properly space memory acknowledgement of the command. When acknowledge is received, SBLD RAISE TR FF H is dropped along with SBLK BUSY L. Coincidentally, SBLK EXPECT RD H is asserted in anticipation of the requested read data.

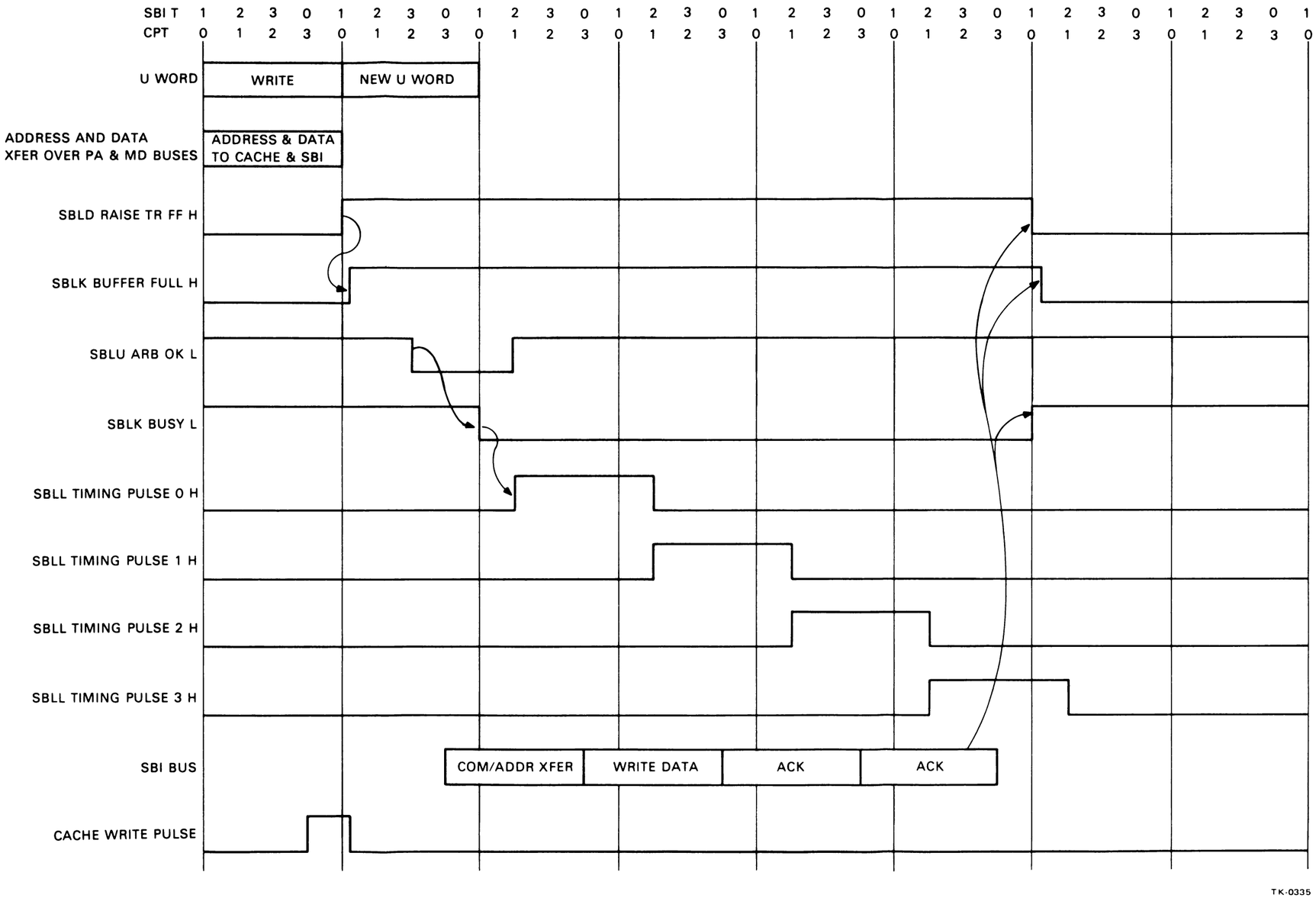
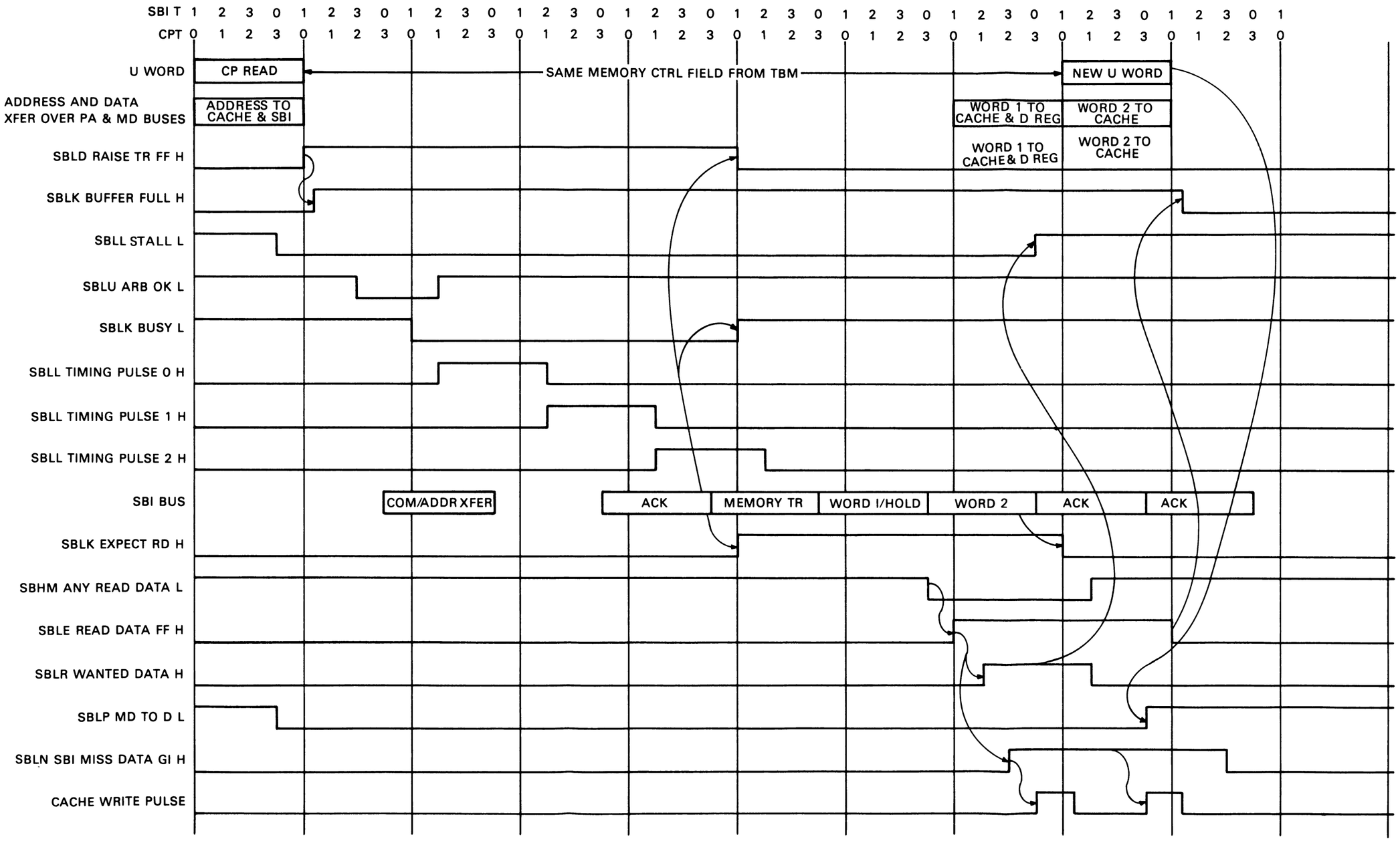


Figure 2-69 Typical SBI Control Write Timing



TK-0360

Figure 2-70 Typical Read Miss Timing

At SBI T0/CPT3 following acknowledgement, memory's transfer request is transmitted. With memory in control of the bus, the first of two data longwords is transmitted from memory to the SBI at SBI T0/CPT3 and latched by the SBI control at SBI T3/CPT2. SBHM ANY READ DATA is generated as a result of SBI decoding. This signal asserts SBLE READ DATA FF H which generates SBLR WANTED DATA H to indicate the requested read data has been received and placed on the MD bus. (In this example, the requested read data is the first of the two longwords.) SBLR WANTED DATA H also unstalls the CPU. SBLP MD TO D L is negated when the requested data is latched from the MD bus by the D register in the data paths. This data is likewise needed for a Cache update and, for this reason, SBLN SBI MISS DATA G1 H is generated. SBLN SBI MISS DATA G1 H enables write pulses to group 1 of the Cache data matrix.

When the second longword is latched from the SBI, SBLK EXPECT RD H is dropped and SBHM ANY READ DATA L is negated. Likewise the unrequested data (indicated by the absence of SBLR WANTED DATA H) is placed on the MD bus for a write to Cache. With SBLN SBI MISS DATA H still asserted, write pulses are again enabled to group 1 of the Cache data matrix.

TRANSLATION BUFFER, CACHE
AND SBI CONTROL TECHNICAL DESCRIPTION
VAX-11/780 IMPLEMENTATION
EK-MM780-TD-001

Reader's Comments

Your comments and suggestions will help us in our continuous effort to improve the quality and usefulness of our publications.

What is your general reaction to this manual? In your judgment is it complete, accurate, well organized, well written, etc.? Is it easy to use? _____

_____

_____

_____

What features are most useful? _____

_____

_____

_____

What faults or errors have you found in the manual? _____

_____

_____

_____

Does this manual satisfy the need you think it was intended to satisfy? _____

Does it satisfy *your* needs? _____ Why? _____

_____

_____

_____

☐ Please send me the current copy of the *Technical Documentation Catalog*, which contains information on the remainder of DIGITAL's technical documentation.

Name _____ Street _____
Title _____ City _____
Company _____ State/Country _____
Department _____ Zip _____

Additional copies of this document are available from:

Digital Equipment Corporation
444 Whitney Street
Northboro, Ma 01532
Attention: Communications Services (NR2/M15)
Customer Services Section

Order No. EK-MM780-TD-001

— — — — — — — — — — — — Fold Here — — — — — — — — — — —

— — — — — — — — — Do Not Tear - Fold Here and Staple — — — — — — — — —

FIRST CLASS
PERMIT NO. 33
MAYNARD, MASS.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

Postage will be paid by:

Digital Equipment Corporation
Technical Documentation Department
Maynard, Massachusetts 01754